Pygame. Введение в разработку игр на Python. Курс

с примерами решений
практических работ

Опубликован: https://younglinux.info/pygame

Автор: Светлана Шапошникова (plustilino)

Версия: ноябрь 2020

Курс "Pygame. Введение в разработку игр на Python" знакомит с базовыми особенностями программирования игр и основными возможностями библиотеки Pygame.

Необходимый предварительный минимум для успешного освоения курса – знание структурного (условия, циклы, функции) и объектно-ориентированного программирования на Python. Желательно хотя бы поверхностное знакомство с программированием приложений с графическим интерфейсом пользователя.

В данном курсе акцент сделан на изучение основных функций и классов библиотеки Pygame, а не на разработку конечных игр. Основные цели курса – познакомиться с особенностями программирования игр и научиться понимать код, использующий Pygame. В последующем это позволит самостоятельно разбирать и понимать длинные примеры написанных на Python и Pygame игр, которых достаточно много в Интернете.

Курс не включает полное описание библиотеки, не дублирует документацию. Вместо этого шаг за шагом, от простого к сложному рассматриваются назначение и принцип работы основных компонентов библиотеки Pygame в приложении к разработке тех или иных элементов компьютерной игры.

Курс состоит из одиннадцати уроков, 10 из которых включают практическую работу. Первый урок-лекция дает общее представление о разработке игр и Pygame.

Урок 1. Pygame и разработка игр

Что такое Pygame

Pygame – это библиотека модулей для языка Python, созданная для разработки 2D игр. Также Pygame могут называть фреймворком. В программировании понятия "библиотека" и "фреймворк" несколько разные. Но когда дело касается классификации конкретного инструмента, не все так однозначно.

В любом случае, фреймворк является более мощным по-сравнению с библиотекой, он накладывает свою специфику на особенности программирования и сферу использования продукта. С точки зрения специфики Pygame – это фреймворк. Однако его сложно назвать "мощным инструментом". По своему объему и функционалу это скорее библиотека.

Также существует понятие "игрового движка" как программной среды для разработки игр. По своему назначению Pygame можно считать игровым движком. В то же время, с точки зрения классификации программного обеспечения, Pygame является API для Питона к API библиотеки SDL.

API – это интерфейс (в основном набор функций и классов) для прикладного (часто более высокоуровневого) программирования, который предоставляет, например, та или иная библиотека. SDL – это библиотека, которая работает с мультимедийными устройствами компьютера.

В этом смысле Pygame можно сравнить с Tkinter, который через свои функции и классы предоставляет Питону доступ к графической библиотеке Tk.

Официальный сайт: https://www.pygame.org

Документация: https://www.pygame.org/docs/.

Особенности разработки компьютерных игр

Игры событийно-ориентированны, также как любое приложение с графическим интерфейсом пользователя. Поэтому какие-никакие, но игры можно было бы писать с помощью Tkinter, в частности на его экземплярах холста. Но поскольку основное назначение библиотеки графического пользовательского интерфейса совсем другое, то пришлось бы изобретать велосипеды. В то время как библиотека, специально предназначенная для написания игр, уже содержит необходимые объекты, что упрощает разработку.

Например, чтобы определить, столкнулись ли два объекта, надо написать код, проверяющий совпадение координат. Это может быть непростой задачей, так как надо учесть области перекрытия, форму объектов и др. В то же время игровой движок может включать готовую функцию проверки коллизии (столкновения) с необходимыми опциями настройки.

При всем этом Pygame достаточно низкоуровневый игровой движок, если его можно так называть. Это значит, что многое в нем не остается за кадром, а дается программисту на доработку, вынуждает его понимать, как работают "шестеренки". Так в Pygame отсутствует эмуляция физических явлений. Если вам надо смоделировать движение с ускорением или по

дуге, программируйте это сами, предварительно взяв из курса физики соответствующую формулу.

Игры относятся к мультимедийным приложениям. Однако, в отличие от других приложений этой группы, для них характерна сложная программная логика и нередко много математики, хотя достаточно простой, плюс эмуляция физических явлений. В играх программируется подобие искусственного интеллекта. В многопользовательской игре, хотя пользователи играют друг с другом, а не с ИИ, создаются виртуальные миры, существующие по законам, заложенным разработчиками.

В программном коде игры выделяют три основных логических блока:

1. Отслеживание событий, производимых пользователем и не только им.
2. Изменение состояний объектов, согласно произошедшим событиям.
3. Отображение объектов на экране, согласно их текущим состояниям.

Эти три этапа повторяются в цикле бесчисленное количество раз, пока игра запущена.

Место Pygame среди инструментов разработки игр

Популярна ли библиотека pygame, пишут ли на ней сложные игры? Хотя на Pygame есть востребованные игры, в подавляющем случае – нет. Для программирования под андроид и десктоп существуют более функциональные игровые движки.

Для создания двумерных браузерных игр инди-разработчики (от слова independent – независимый, здесь понимается как "одиночка", "не работающий в команде или на фирму") часто используют JavaScript и его игровые библиотеки, так как JS родной для веба язык. Хотя существуют проекты перевода с Python на JavaScript (https://github.com/jggatc/pyjsdl).

Для запуска python-приложений на Android см. https://github.com/kivy/python-for-android и https://github.com/duducosmos/pgs4a.

В чем тогда преимущество Pygame? Оно в легком вхождении в отрасль и прототипировании. Pygame – небольшая библиотека. Сам Python позволяет писать короткий и ясный код. Так что это 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 программа переписывается на другом языке. Другими словами, преимущество Pygame в легком обучении и быстрой разработке.

После Pygame жизнь разработчика игр на Питоне не заканчивается. Следует посмотреть в сторону Kivy (https://kivy.org). Это уже полноценный фреймворк, позволяющий писать на Python не только игровые приложения. В большей степени ориентирован для разработки под мобильные платформы.

Как установить Pygame

Pygame не входит в стандартную библиотеку Python, то есть не поставляется с установочным пакетом, а требует отдельной установки. В Ubuntu и родственных дистрибутивах его можно установить с помощью pip:

```
python3 -m pip install -U pygame --user
```

Если pip не установлен, предварительно выполняем команду:

```
sudo apt install python3-pip
```

Для Windows:

```
py -m pip install -U pygame --user
```

Дополнительную информацию по установке смотрите здесь:
https://www.pygame.org/wiki/GettingStarted

Проверить, что все установилось нормально, можно так:

```
python3 -m pygame.examples.aliens
```

Для Windows вместо 'python3' надо писать 'py'. Произойдет запуск игры aliens, включенной в модуль examples (примеры) библиотеки pygame.

Урок 2. Каркас игры на Pygame

Pygame задает особые правила построения кода. Эти правила не являются строгими. Однако в большинстве случаев, чтобы игра благополучно запустилась, в программе должна быть соблюдена определенная последовательность вызова ключевых команд.

Эти команды (импорт модуля, вызовы функций, цикл) создают своего рода скелет, или каркас, программного кода. Выполнив его, вы получите "пустую" игру. Далее на этот скелет "подвешивается мясо", т. е. объявляются объекты и программируется логика игры.

Первое, что нужно сделать, это импортировать модуль pygame. После этого можно вывести на экран главное графическое окно игры с помощью функции set_mode() модуля display, входящего в состав библиотеки pygame:

```
import pygame

pygame.display.set_mode((600, 400))
```

Если выполнить этот код, то появится окно размером 600x400 пикселей и сразу закроется (в Linux, в Windows может зависнуть).

Функция set_mode() принимает три аргумента – размер в виде кортежа из двух целых чисел, флаги и глубину цвета. Их можно не указывать. В этом случае окно займет весь экран, цветовая глубина будет соответствовать системной. Обычно указывают только первый аргумент – размер окна.

Флаги предназначены для переключения на аппаратное ускорение, полноэкранный режим, отключения рамки окна и др. Например, команда `pygame.display.set_mode((640, 560), pygame.RESIZABLE)` делает окно изменяемым в размерах.

Выражение вида pygame.RESIZABLE (вместо RESIZABLE может быть любое другое слово большими буквами) обозначает обращение к той или иной константе, определенной в модуле pygame. Часто можно встретить код, в котором перед константами не пишется имя модуля (вместо, например, pygame.QUIT пишут просто QUIT). В этом случае в начале программы надо импортировать не только pygame, но и содержимое модуля locals через from … import:

```
import pygame
from pygame.locals import *
```

Однако в данном курсе мы оставим длинное обращение к встроенным константам, чтобы на этапе обучения не путать определенные в модуле и свои собственные, которые нам также придется создавать.

Функция set_mode() возвращает объект типа Surface (поверхность). В программе может быть множество объектов данного класса, но тот, что возвращает set_mode() особенный. Его называют display surface, что можно перевести как экранная (дисплейная) поверхность. Она главная.

В конечном итоге все отображается на ней с помощью функции pygame.display.update() или родственной pygame.display.flip(), и именно эту поверхность мы видим на экране монитора.

Нам пока нечего отображать, мы не создавали никаких объектов. Поэтому было показано черное окно.

Функции update() и flip() модуля display обновляют содержимое окна игры. Это значит, что каждому пикселю заново устанавливается цвет. Представьте, что на зеленом фоне движется красный круг. За один кадр круг смещается на 5 пикселей. От кадра к кадру картинка целого окна изменяется незначительно, но в памяти окно будет перерисовываться полностью. Если частота составляет 60 кадров в секунду (FPS=60), то за секунду в памяти компьютера произойдет 60 обновлений множества значений, соответствующих экранным пикселям, что дает по большей части бессмысленную нагрузку на вычислительные мощности.

Если функции update() не передавать аргументы, то будут обновляться значения всей поверхности окна. Однако можно передать более мелкую прямоугольную область или список таковых. В этом случае обновляться будут только они.

Функция flip() решает проблему иным способом. Она дает выигрыш, если в set_mod() были переданы определенные флаги (аппаратное ускорение + полноэкранный режим – pygame.HWSERFACE|pygame.FULLSCREEN, двойная буферизация – pygame.DOUBLEBUFF, использование OpenGL – pygame.OPENGL). Возможно, все флаги можно комбинировать вместе (через |). При этом, согласно документации, аппаратное ускорение работает только в полноэкранном режиме.

Вернемся к нашим трем строчкам кода. Почему окно сразу закрывается? Очевидно потому, что программа заканчивается после выполнения этих выражений. Ни init(), ни set_mode() не предполагают входа в "режим циклического ожидания событий". В tkinter для этого используется метод mainloop() экземпляра Tk(). В pygame же требуется собственноручно создать бесконечный цикл, заставляющий программу зависнуть. Основная причина в том, что только программист знает, какая часть его кода должна циклически обрабатываться, а какая – нет. Например, код, создающий классы, объекты и функции не "кладут" в цикл.

Итак, создадим в программе бесконечный цикл:

```
#  Осторожно! Эта программа зависнет.
import pygame as pg

pg.display.set_mode((600, 400))

while 1:
    pass
```

После такого окно уже не закроется, а программа благополучно зависнет насовсем. Многократные клики по крестику не помогут. Только принудительная остановка программы через среду разработки или Ctrl+C, если запускали через терминал.

Как сделать так, чтобы программа закрывалась при клике на крестик окна, а также при нажатии Alt+F4? Pygame должен воспринимать такие действия как определенный тип событий.

Добавим в цикл магии:

```
#  Окно закроется, но с ошибкой.
import pygame as pg

pg.display.set_mode((600, 400))

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            pg.quit()
```

При выходе будет генерироваться ошибка, пока забудем про нее. Сейчас достаточно того, что окно успешно закрывается.

Рассмотрим выражение pygame.event.get(). Модуль event библиотеки pygame содержит функцию get(), которая забирает список событий из очереди, в которую записываются все произошедшие события. То, что возвращает get() – это список. Забранные события удаляются из очереди, то есть второй раз они уже забираться не будут, а в очередь продолжают записываться новые события.

Цикл for просто перебирает схваченный на данный момент (в текущей итерации цикла) список событий. Каждое событие он присваивает переменной i или любой другой. Чтобы было понятней, перепишем программу таким образом:

```
#  Окно закроется, но с ошибкой.
import pygame as pg

pg.display.set_mode((600, 400))

while 1:
    events = pg.event.get()
    print(events)
    for i in events:
        if i.type == pg.QUIT:
            print(pg.QUIT)
            print(i)
            print(i.type)
            pg.quit()
```

На экране вы увидите примерно такое:

```
…
[]
[<Event(512-...)>, <Event(256-Quit {})>]
256
<Event(256-Quit {})>
256
Traceback (most recent call last):
  File "/home/pl/…/event2.py", line 7, in <module>
    events = pg.event.get()
pygame.error: video system not initialized
```

Вверху будет множество пустых квадратных скобок, которые соответствуют пустым спискам events, создаваемым на каждой итерации цикла while. И только когда окно закрывается, генерируются два события. Свойство type второго имеет значение 256, что совпадает со значением константы QUIT.

В pygame событие – это объект класса Event. А если это объект, то у него есть атрибуты (свойства и методы). В данном случае мы отслеживаем только те события, у которых значение свойства type совпадает со значением константы QUIT модуля pygame. Это значение присваивается type тогда, когда происходят события нажатия на крестик или Alt+F4. Когда эти события происходят, то в данном случае мы хотим, чтобы выполнилась функция quit() модуля pygame, которая завершает его работу.

Теперь почему возникает ошибка. Функция pygame.quit() отключает (деинициализирует) pygame, но не завершает работу программы. Таким образом, после выполнения этой функции отключаются модули библиотеки pygame, но выхода из цикла и программы не происходит. Программа продолжает работу и переходит к следующей итерации цикла while (или продолжает выполнять тело данной итерации, если оно еще не закончилось).

В данном случае происходит переход к следующей итерации цикла while. И здесь выполнить функцию get() модуля event оказывается уже невозможным. Возникает исключение и программа завершается. По-сути программу завершает не функция pygame.quit(), а выброшенное, но не обработанное, исключение.

Данную проблему можно решить разными способами. Часто используют функцию exit() модуля sys. В этом случае код выглядит примерно так:

```
import pygame as pg
import sys

pg.display.set_mode((600, 400))

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            pg.quit()
            sys.exit()
```

Сначала отключается pygame, потом происходит выход из программы. Такой вариант вероятно следует считать наиболее безопасным завершением. Команда pygame.quit() не обязательна. Если завершается программа, то отключится и pygame.

Другой вариант – не допустить следующей итерации цикла. Для этого потребуется дополнительная переменная:

```
import pygame as pg

pg.display.set_mode((600, 400))

play = True
while play:
```

```
    for i in pg.event.get():
        if i.type == pg.QUIT:
            play = False
```

В этом случае завершится текущая итерация цикла, но новая уже не начнется. Если в основной ветке ниже по течению нет другого кода, программа завершит свою работу.

Нередко код основной ветки программы помещают в функцию, например, main(). Она выполняется, если файл запускается как скрипт, а не импортируется как модуль. В этом случае для завершения программы проще использовать оператор return, который осуществляет выход из функции.

```
import pygame as pg


def main():
    pg.display.set_mode((600, 400))

    while True:
        for i in pg.event.get():
            if i.type == pg.QUIT:
                return


if __name__ == "__main__":
    main()
```

Теперь зададимся вопросом, с какой скоростью крутится цикл while? С большой, зависящей от мощности компьютера. Но в данном случае такая скорость не есть необходимость, она даже вредна, так как бессмысленно расходует ресурсы. Человек дает команды и воспринимает изменения куда медленнее.

Для обновления экрана в динамической игре часто используют 60 кадров в секунду, а в статической, типа пазла, достаточно будет 30-ти. Из этого следует, что циклу незачем работать быстрее.

Поэтому в главном цикле следует выполнять задержку. Делают это либо вызовом функции delay() модуля time библиотеки pygame, либо создают объект часов и устанавливают ему частоту кадров. Первый способ проще, второй – более профессиональный.

```
...
while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
    pg.time.delay(20)
```

Функция delay() принимает количество миллисекунд (1000 мс = 1 с). Если передано значение 20, то за секунду экран обновится 50 раз. Другими словами, частота составит 50 кадров в секунду.

```
import pygame as pg
import sys

pg.display.set_mode((600, 400))

clock = pg.time.Clock()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
    clock.tick(60)
```

Методу tick() класса Clock передается непосредственно желаемое количество кадров в секунду. Задержку он вычисляет сам. То есть если внутри цикла указано tick(60) – это не значит, что задержка будет 60 миллисекунд или произойдет 60 обновлений экрана за одну итерацию цикла. Это значит, что на каждой итерации цикла секунда делится на 60 и уже на вычисленную величину выполняется задержка.

Нередко частоту кадров выносят в отдельную константоподобную переменную:

```
...
FPS = 60
...
clock = pg.time.Clock()

while 1:
    ...
    clock.tick(FPS)
```

В начало цикла или конец вставлять задержку зависит от контекста. Если до цикла происходит отображение каких-либо объектов на экране, то скорее всего надо вставлять в начало цикла. Если первое появление объектов на экране происходит внутри цикла, то в конец.

В итоге каркас игры на Pygame должен выглядеть примерно так:

```
# здесь подключаются модули
import pygame
import sys

# здесь определяются константы,
# классы и функции
FPS = 60

# здесь происходит инициация,
# создание объектов
pygame.init()
pygame.display.set_mode((600, 400))
clock = pygame.time.Clock()

# если надо до цикла отобразить
```

```
# какие-то объекты, обновляем экран
pygame.display.update()

# главный цикл
while True:

    # задержка
    clock.tick(FPS)

    # цикл обработки событий
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()

    # --------
    # изменение объектов
    # --------

    # обновление экрана
    pygame.display.update()
```

Практическая работа

В модуле pygame.display есть функция set_caption(). Ей передается строка, которую она устанавливает в качестве заголовка окна. Сделайте так, чтобы каждую секунду заголовок окна изменялся.

Урок 3. Модуль pygame.draw – геометрические примитивы

Функции модуля pygame.draw рисуют геометрические примитивы на поверхности – экземпляре класса Surface. В качестве первого аргумента они принимают поверхность. Поэтому при создании той или иной поверхности ее надо связать с переменной, чтобы потом было что передать в функции модуля draw. Поскольку мы пока используем только одну поверхность – главную оконную, то ее будем указывать в качестве первого параметра, а при создании свяжем с переменной:

```
import pygame as pg
import sys

sc = pg.display.set_mode((300, 200))

# здесь будут рисоваться фигуры

pg.display.update()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
    pg.time.delay(1000)
```

В большинстве случаев фигуры прорисовывают внутри главного цикла, так как от кадра к кадру картинка на экране должна меняться. Поэтому на каждой итерации цикла в функции модуля draw передаются измененные аргументы (например, каждый раз меняется координата *x*).

Однако у нас пока не будет никакой анимации, и нет смысла перерисовывать фигуры на одном и том же месте на каждой итерации цикла. Поэтому создавать примитивы будем в основной ветке программы. На данном этапе цикл while нужен лишь для того, чтобы программа самопроизвольно не завершалась.

После прорисовки, чтобы увидеть изменения в окне игры, необходимо выполнить функцию update() или flip() модуля display. Иначе окно не обновится. Рисование на поверхности – одно, а обновление состояния главного окна – другое. Представьте, что в разных местах тела главного цикла на поверхности прорисовываются разные объекты. Если бы каждое такое действие приводило к автоматическому обновлению окна, то за одну итерацию оно обновлялось бы несколько раз. Это приводило бы как минимум к бессмысленной трате ресурсов, так как скорость цикла связана с FPS.

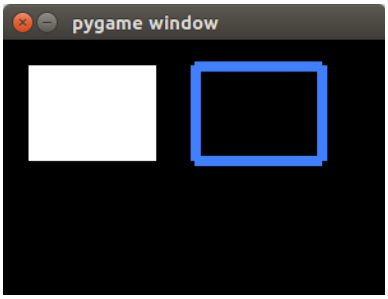
Итак, первый аргумент функций рисования – поверхность, на которой размещается фигура. В нашем случае это будет `sc`. Вторым обязательным аргументом является цвет. Цвет задается в формате RGB, используется трехэлементный целочисленный кортеж. Например, (255, 0, 0) определяет красный цвет.

Далее идут специфичные для каждой фигуры аргументы. Последним у большинства является толщина контура.

Все функции модуля draw возвращают экземпляры класса Rect – прямоугольные области, имеющие координаты, длину и ширину. Не путайте функцию rect() модуля draw и класс Rect, это разные вещи.

Начнем с функции rect() модуля draw:

```
pygame.draw.rect(sc, (255, 255, 255),
                 (20, 20, 100, 75))
pygame.draw.rect(sc, (64, 128, 255),
                 (150, 20, 100, 75), 8)
```



Если указывается толщина контура (последний аргумент во второй строке), то прямоугольник будет незаполненным, а цвет определит цвет рамки. Третьим аргументом является кортеж из четырех чисел. Первые два определяют координаты верхнего левого угла прямоугольника, вторые – его ширину и высоту.

Следует отметить, что в функцию draw.rect() и некоторые другие третьим аргументом можно передавать не кортеж, а заранее созданный экземпляр Rect. В примере ниже показан такой вариант.

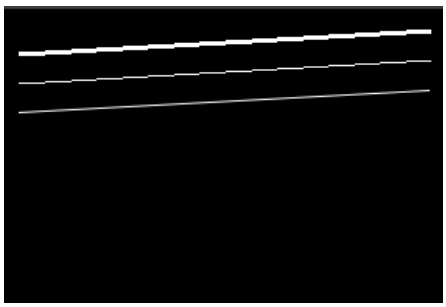
Обычно цвета выносят в отдельные переменные-константы. Это облегчает чтение кода:

```
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GRAY = (125, 125, 125)
LIGHT_BLUE = (64, 128, 255)
GREEN = (0, 200, 64)
YELLOW = (225, 225, 0)
PINK = (230, 50, 230)

r1 = pygame.Rect((150, 20, 100, 75))

pygame.draw.rect(sc, WHITE, (20, 20, 100, 75))
pygame.draw.rect(sc, LIGHT_BLUE, r1, 8)
```

Чтобы нарисовать линию, а точнее – отрезок, надо указать координаты его концов. При этом функция line() рисует обычную линию, aaline() – сглаженную (толщину для последней указать нельзя):

```
pygame.draw.line(sc, WHITE,
                 [10, 30],
                 [290, 15], 3)
```

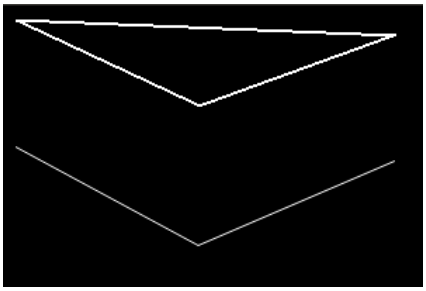
```
pygame.draw.line(sc, WHITE,
                 [10, 50],
                 [290, 35])
pygame.draw.aaline(sc, WHITE,
                   [10, 70],
                   [290, 55])
```



Координаты можно передавать как в виде списка, так и кортежа.

Функции lines() и aalines() рисуют ломанные линии:

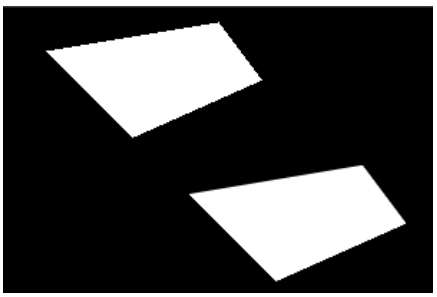
```
pygame.draw.lines(sc, WHITE, True,
                  [[10, 10], [140, 70],
                   [280, 20]], 2)
pygame.draw.aalines(sc, WHITE, False,
                    [[10, 100], [140, 170],
                     [280, 110]])
```



Координаты определяют места излома. Количество точек может быть произвольным. Третий параметр (True или False) указывает замыкать ли крайние точки.

Функция polygon() рисует произвольный многоугольник. Задаются координаты вершин.

```
pygame.draw.polygon(sc, WHITE,
                    [[150, 10], [180, 50],
                     [90, 90], [30, 30]])
pygame.draw.polygon(sc, WHITE,
                    [[250, 110], [280, 150],
                     [190, 190], [130, 130]])
pygame.draw.aalines(sc, WHITE, True,
                    [[250, 110], [280, 150],
                     [190, 190], [130, 130]])
```

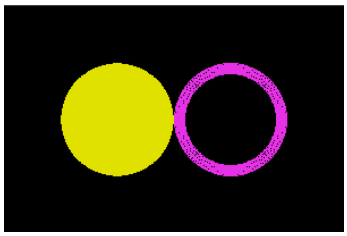


Сглаженная ломаная здесь повторяет контур многоугольника, чем сглаживает его ребра.

Так же как в случае rect() для polygon() можно указать толщину контура.

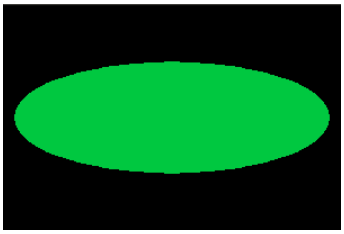
Функция circle() рисует круги. Указывается центр окружности и радиус:

```
pygame.draw.circle(sc, YELLOW,
                   (100, 100), 50)
pygame.draw.circle(sc, PINK,
                   (200, 100), 50, 10)
```



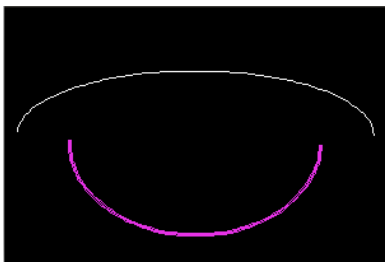
В случае эллипса передается описывающая его прямоугольная область:

```
pygame.draw.ellipse(sc, GREEN,
                    (10, 50, 280, 100))
```



Наконец, дуга:

```
pi = 3.14
pygame.draw.arc(sc, WHITE,
                (10, 50, 280, 100),
                0, pi)
pygame.draw.arc(sc, PINK,
                (50, 30, 200, 150),
                pi, 2*pi, 3)
```



Указывается прямоугольник, описывающий эллипс, из которого вырезается дуга. Четвертый и пятый аргументы – начало и конец дуги, выраженные в радианах. Нулевая точка справа.

Практическая работа. Анимация

На данном этапе мы уже готовы создать анимацию. Никакого движения объектов на экране монитора нет. Просто от кадра к кадру изменяются цвета пикселей экрана. Например, пиксель с координатами (10, 10) светится синим цветом, в следующем кадре синим загорается пиксель (11, 11), в то время как (10, 10) становится таким же как фон. В следующем кадре синей будет только точка (12, 12) и так далее. При этом человеку будет казаться, что синяя точка движется по экрану по диагонали.

Суть алгоритма в следующем. Берем фигуру. Рисуем ее на поверхности. Обновляем главное окно, человек видит картинку. Стираем фигуру. Рисуем ее с небольшим смещением от первоначальной позиции. Снова обновляем окно и так далее.

Как "стереть" старую фигуру? Для этого используется метод fill() объекта Surface. В качестве аргумента передается цвет, т. е. фон можно сделать любым, а не только черным, который задан по-умолчанию.

Ниже в качестве примера приводится код анимации круга. Объект появляется с левой стороны, доходит до правой, исчезает за ней. После этого снова появляется слева. Ваша задача написать код анимации квадрата, который перемещается от левой границе к правой, касается ее, но не исчезает за ней. После этого возвращается назад – от правой границы к левой, касается ее, опять двигается вправо. Циклы движения квадрата повторяются до завершения программы.

```
import pygame
import sys

FPS = 60
WIN_WIDTH = 400
WIN_HEIGHT = 100
WHITE = (255, 255, 255)
ORANGE = (255, 150, 100)

clock = pygame.time.Clock()
sc = pygame.display.set_mode(
    (WIN_WIDTH, WIN_HEIGHT))

# радиус будущего круга
```

```
r = 30
# координаты круга
# скрываем за левой границей
x = 0 - r
# выравнивание по центру по вертикали
y = WIN_HEIGHT // 2

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()

    # заливаем фон
    sc.fill(WHITE)
    # рисуем круг
    pygame.draw.circle(sc, ORANGE,
                       (x, y), r)
    # обновляем окно
    pygame.display.update()

    # Если круг полностью скрылся
    # за правой границей,
    if x >= WIN_WIDTH + r:
        # перемещаем его за левую
        x = 0 - r
    else:  # Если еще нет,
        # на следующей итерации цикла
        # круг отобразится немного правее
        x += 2

    clock.tick(FPS)
```

Урок 4. События клавиатуры

Человек может управлять объектами в игре в основном с помощь клавиатуры, мыши, джойстика. Когда на "манипуляторах" что-то двигается или нажимается, то возникают события определенных типов. Обработкой событий занимается модуль pygame.event, который включает ряд функций, наиболее важная из которых уже ранее рассмотренная pygame.event.get(), которая забирает из очереди произошедшие события.

В pygame, когда фиксируется то или иное событие, создается соответствующий ему объект от класса Event. Уже с этими объектами работает программа. Экземпляры данного класса имеют только свойства, у них нет методов. У всех экземпляров есть свойство type. Набор остальных свойств события зависит от значения type.

События клавиатуры могут быть двух типов (иметь одно из двух значений type) – клавиша была нажата, клавиша была отпущена. Если вы нажали клавишу и отпустили, то в очередь событий будут записаны оба. Какое из них обрабатывать, зависит от контекста игры. Если вы зажали клавишу и не отпускаете ее, то в очередь записывается только один вариант – клавиша нажата.

Событию типа "клавиша нажата" в поле type записывается числовое значение, совпадающее со значением константы pygame.KEYDOWN. Событию типа "клавиша отпущена" в поле type записывается значение, совпадающее со значением константы pygame.KEYUP.

У обоих типов событий клавиатуры есть атрибуты key и mod. В key записывается конкретная клавиша, которая была нажата или отжата. В mod – клавиши-модификаторы (Shift, Ctrl и др.), которые были зажаты в момент нажатия или отжатия обычной клавиши. У событий KEYDOWN также есть поле unicode, куда записывается символ нажатой клавиши (тип данных str).

Рассмотрим, как это работает. Пусть в центре окна имеется круг, который можно двигать по горизонтали клавишами стрелок клавиатуры:

```
import pygame
import sys

FPS = 60
W = 700  # ширина экрана
H = 300  # высота экрана
WHITE = (255, 255, 255)
BLUE = (0, 70, 225)

sc = pygame.display.set_mode((W, H))
clock = pygame.time.Clock()

# координаты и радиус круга
x = W // 2
y = H // 2
r = 50
```

```
while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYDOWN:
            if i.key == pygame.K_LEFT:
                x -= 3
            elif i.key == pygame.K_RIGHT:
                x += 3

    sc.fill(WHITE)
    pygame.draw.circle(sc, BLUE, (x, y), r)
    pygame.display.update()
    clock.tick(FPS)
```

В цикле обработки событий теперь проверяется не только событие выхода, но также нажатие клавиш. Сначала необходимо проверить тип, потому что не у всех событий есть атрибут key. Если сразу начать проверять key, то сгенерируется ошибка по той причине, что могло произойти множество событий. Например, движение мыши, у которого нет поля key. Соответственно, попытка взять значение из несуществующего поля (i.key) приведет к генерации исключения.

Часто проверку и типа и клавиши записывают в одно логическое выражение (`i.type == pygame.KEYDOWN and i.key == pygame.K_LEFT`). В Python так можно делать потому, что если первая часть сложного выражения возвращает ложь, то вторая часть уже не проверяется.

Если какая-либо клавиша была нажата, то проверяется, какая именно. В данном случае обрабатываются только две клавиши. В зависимости от этого меняется значение координаты *x*.

Проблема данного кода в том, что при выполнении программы, чтобы круг двигался, надо постоянно нажимать и отжимать клавиши. Если просто зажать их на длительный период, то объект не будет постоянно двигаться. Он сместиться только одноразово на 3 пикселя.

Так происходит потому, что событие нажатия на клавишу происходит один раз, сколь долго бы ее не держали. Это событие было забрано из очереди функцией get() и обработано. Его больше нет. Поэтому приходится генерировать новое событие, еще раз нажимая на клавишу.

Как быть, если по логике вещей надо, чтобы шар двигался до тех пор, пока клавиша зажата? Когда же она отпускается, шар должен останавливаться. Первое, что надо сделать, – это перенести изменение координаты *x* в основную ветку главного цикла while. В таком случае на каждой его итерации координата будет меняться, а значит шар двигаться постоянно.

Во-вторых, в цикле обработки событий нам придется следить не только за нажатием клавиши, но и ее отжатием. Когда клавиша нажимается, какая-либо переменная, играющая роль флага, должна принимать одно значение, когда клавиша отпускается эта же переменная должна принимать другое значение.

В основном теле while надо проверять значение этой переменной и в зависимости от него менять или не менять значение координаты.

```
import pygame
import sys

FPS = 60
W = 700  # ширина экрана
H = 300  # высота экрана
WHITE = (255, 255, 255)
BLUE = (0, 70, 225)
RIGHT = "to the right"
LEFT = "to the left"
STOP = "stop"

sc = pygame.display.set_mode((W, H))
clock = pygame.time.Clock()

# координаты и радиус круга
x = W // 2
y = H // 2
r = 50

motion = STOP

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYDOWN:
            if i.key == pygame.K_LEFT:
                motion = LEFT
            elif i.key == pygame.K_RIGHT:
                motion = RIGHT
        elif i.type == pygame.KEYUP:
            if i.key in [pygame.K_LEFT,
                         pygame.K_RIGHT]:
                motion = STOP

    sc.fill(WHITE)
    pygame.draw.circle(sc, BLUE, (x, y), r)
    pygame.display.update()

    if motion == LEFT:
        x -= 3
    elif motion == RIGHT:
        x += 3

    clock.tick(FPS)
```

Использовать константы не обязательно, можно сразу присваивать строки или даже числа (например, motion = 1 обозначает движение вправо, -1 – влево, 0 – остановка). Однако константы позволяют легче понимать и обслуживать в дальнейшем код, делают его более информативным. Лучше привыкнуть к такому стилю.

Должно проверяться отжатие только двух клавиш. Если проверять исключительно KEYUP без последующей конкретизации, то отжатие любой клавиши приведет к остановке, даже если в это время будет по-прежнему зажиматься клавиша влево или вправо. Выражение `i.key in [pygame.K_LEFT, pygame.K_RIGHT]` обозначает, что если значение i.key совпадает с одним из значений в списке, то все выражение возвращает истину.

На самом деле существует способ по-проще. В библиотеке pygame с событиями работает не только модуль event. Так модуль pygame.key включает функции, связанные исключительно с клавиатурой. Здесь есть функция pygame.key.get_pressed(), которая возвращает кортеж двоичных значений. Индекс каждого значения соответствует своей клавиатурной константе. Само значение равно 1, если клавиша нажата, и 0 – если не нажата.

Эта функция подходит не для всех случаев обработки клавиатурных событий, но в нашем подойдет. Поэтому мы можем упростить код до такого варианта:

```
import pygame
import sys

FPS = 60
W = 700  # ширина экрана
H = 300  # высота экрана
WHITE = (255, 255, 255)
BLUE = (0, 70, 225)

sc = pygame.display.set_mode((W, H))
clock = pygame.time.Clock()

# координаты и радиус круга
x = W // 2
y = H // 2
r = 50

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()

    sc.fill(WHITE)
    pygame.draw.circle(sc, BLUE, (x, y), r)
    pygame.display.update()

    keys = pygame.key.get_pressed()

    if keys[pygame.K_LEFT]:
        x -= 3
    elif keys[pygame.K_RIGHT]:
        x += 3

    clock.tick(FPS)
```

Можно сказать, вызов get_pressed() снимает "маску" зажатых клавиш. Мы ее снимаем на каждой итерации главного цикла. Это даже не регистрация событий как таковых.

Выражение типа keys[pygame.K_LEFT] извлекает значение из кортежа по индексу, значение которого записано в константе K_LEFT. Если извлеченное значение True, то координата меняется.

Если необходимо, чтобы событие обрабатывалось при нажатии двух и более клавиш, то работает такое логическое выражение: `keys[pygame.K_LEFT] and keys[pygame.K_a]` (одновременное нажатие стрелки 'влево' и буквы 'a'). Однако если нужно задействовать не обычные клавиши, а модификаторы, то данный номер не проходит.

В таком случае можно вернуться к первому варианту – перебирать события в цикле for:

```
...
        elif i.type == pygame.KEYDOWN:
            if i.key == pygame.K_LEFT and\
                    (i.mod & pygame.KMOD_SHIFT):
                motion = LEFT
...
```

Здесь при if будет True, если перед нажатием стрелки был зажат левый Shift. Причем обратная последовательность: сначала зажать стрелку, потом Shift не сработает. Видимо модификаторы обрабатываются библиотекой pygame несколько отлично от обычных клавиш. Допустим, если при зажатии обычных клавиш генерируется только одно событие, то для модификаторов они генерируются постоянно или хранятся до отпускания в другой очереди.

Таким образом, если первым зажимается K_LEFT, то событие сразу обрабатывается. При этом в i.mod записывается отсутствие модификатора. Поэтому условие не срабатывает.

Если же первым зажимается модификатор, то это событие не теряется и позволяет условию при if выполнится в случае нажатия при этом обычной клавиши.

Весь перечень констант pygame, соответствующих клавишам клавиатуры, смотрите в документации: https://www.pygame.org/docs/ref/key.html

Практическая работа

Измените приведенную в уроке программу так, чтобы круг с той же скоростью, т. е. постепенно, возвращался назад в исходную точку, когда клавиша отпускается.

Урок 5. События мыши

В Pygame обрабатываются три типа событий мыши:

- нажатие кнопки (значение свойства **type** события соответствует константе pygame.MOUSEBUTTONDOWN),
- отпускание кнопки (MOUSEBUTTONUP),
- перемещение мыши (MOUSEMOTION).

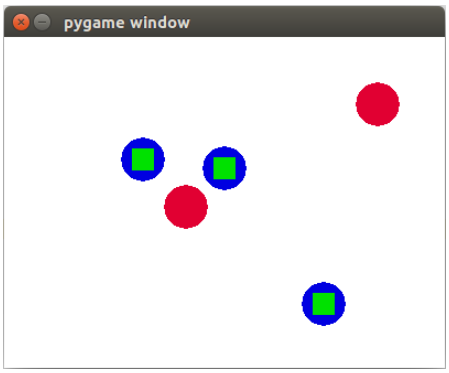
Какая именно кнопка была нажата, записывается в другое свойство события – **button**. Для левой кнопки это число 1, для средней – 2, для правой – 3, для прокручивания вперед – 4, для прокручивания назад – 5. У событий MOUSEMOTION вместо button используется свойство buttons, в которое записывается состояние трех кнопок мыши (кортеж из трех элементов).

Другим атрибутом мышиных типов событий является свойство **pos**, в которое записываются координаты происшествия (кортеж из двух чисел).

Таким образом, если вы нажали правую кнопку мыши точно в середине окна размером 200x200, то будет создан объект типа Event с полями event.type = pygame.MOUSEBUTTONDOWN, event.button = 3, event.pos = (100, 100).

У событий MOUSEMOTION есть еще один атрибут – **rel**. Он показывает относительное смещение по обоим осям. С помощью него, например, можно отслеживать скорость движения мыши.

Код ниже создает фигуры в местах клика мыши. Нажатие средней кнопки очищает поверхность.




```
import pygame as pg
import sys

WHITE = (255, 255, 255)
RED = (225, 0, 50)
GREEN = (0, 225, 0)
BLUE = (0, 0, 225)

sc = pg.display.set_mode((400, 300))
sc.fill(WHITE)
```

```
pg.display.update()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
        if i.type == pg.MOUSEBUTTONDOWN:
            if i.button == 1:
                pg.draw.circle(
                    sc, RED, i.pos, 20)
                pg.display.update()
            elif i.button == 3:
                pg.draw.circle(
                    sc, BLUE, i.pos, 20)
                pg.draw.rect(
                    sc, GREEN,
                    (i.pos[0] - 10,
                     i.pos[1] - 10,
                     20, 20))
                pg.display.update()
            elif i.button == 2:
                sc.fill(WHITE)
                pg.display.update()

    pg.time.delay(20)
```

В функции модуля draw вместо координат передается значение поля pos события. В pos хранятся координаты клика. В случае с функцией rect() извлекаются отдельные элементы кортежа pos. Вычитание числа 10 используется для того, чтобы середина квадрата, сторона которого равна 20-ти пикселям, точно соответствовала месту клика. Иначе в месте клика будет находиться верхний левый угол квадрата.

Функцию update() не обязательно вызывать три раза в ветках if-elif-elif. Ее можно вызвать в основном теле главного цикла. Однако в этом случае, когда кликов не происходит, она будет выполнять зря.

Также как в случае с клавиатурой в pygame есть свой модуль для событий мыши. Если нужно отслеживать длительное зажатие ее кнопок, следует воспользоваться функцией get_pressed() модуля pygame.mouse. Здесь же есть функция для считывания позиции курсора – get_pos(). Следующий код рисует синий круг в местах клика левой кнопкой мыши:

```
import pygame as pg
import sys

WHITE = (255, 255, 255)
BLUE = (0, 0, 225)

sc = pg.display.set_mode((400, 300))
sc.fill(WHITE)
pg.display.update()
```

```
while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()

    pressed = pg.mouse.get_pressed()
    pos = pg.mouse.get_pos()
    if pressed[0]:
        pg.draw.circle(sc, BLUE, pos, 5)
        pg.display.update()

    pg.time.delay(20)
```

Функция mouse.get_pressed() возвращает трехэлементный кортеж. Первый элемент (с индексом 0) соответствует левой кнопке мыши, второй – средней, третий – правой. Если значение элемента равно True, значит, кнопка нажата. Если False, значит – нет. Так выражение pressed[0] есть истина, если под нулевым индексом содержится True.

Чтобы скрыть курсор (например, в игре, где управление осуществляется исключительно клавиатурой), надо воспользоваться функцией pygame.mouse.set_visible(), передав в качестве аргумента False.

Так можно привязать графический объект к курсору (в данном случае привязывается квадрат):

```
import pygame as pg
import sys

WHITE = (255, 255, 255)
BLUE = (0, 0, 225)

pg.init()
sc = pg.display.set_mode((400, 300))
sc.fill(WHITE)
pg.display.update()

pg.mouse.set_visible(False)

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()

    sc.fill(WHITE)

    if pg.mouse.get_focused():
        pos = pg.mouse.get_pos()
        pg.draw.rect(
            sc, BLUE, (pos[0] - 10,
                       pos[1] - 10,
                       20, 20))
```

```
    pg.display.update()
    pg.time.delay(20)
```

Функцией get_pos() мы можем считывать позицию курсора, даже если он не виден. Далее в этой позиции рисуем фигуру на каждой итерации цикла.

Функция get_focused() проверяет, находится ли курсор в фокусе окна игры. Если не делать эту проверку, то при выходе курсора за пределы окна, квадрат будет постоянно прорисовываться у края окна, где произошел выход, т. е. не будет исчезать.

Практическая работа

Напишите код в котором имитируется полет снаряда (пусть его роль сыграет круг) в место клика мышью. Снаряд должен вылетать из нижнего края окна и лететь вверх, т. е. изменяться должна только координата *y*. Пока летит один, другой не должен появляться. Когда снаряд достигает цели, должен имитировать взрыв, например, в этом месте прорисовываться квадрат.

Урок 6. Класс Surface и метод blit()

С помощью класса pygame.Surface можно создавать дополнительные поверхности. После этого отрисовывать их на основной, которая создается методом pygame.display.set_mode(), или друг на друге. Отрисовка выполняется с помощью метода blit().

В pygame поверхности создаются не только вызовом функции display.set_mode() или напрямую вызовом конструктора класса Surface. Также в результате выполнения ряда других функций и методов. Это связано с тем, что поверхности играют важную роль, так как в конечном итоге именно они отображаются на экране. Кроме того они позволяют группировать объекты. Их можно сравнить со слоями в анимации.

При создании экземпляра Surface непосредственно от класса необходимо указать ширину и высоту, подобно тому, как это происходит при вызове set_mode(). Например:

```
surf = pygame.Surface((150, 150))
```

Метод blit() применяется к той поверхности, на которую "накладывается", т. е. на которой "отрисовывается", другая. Другими словами, метод blit() применяется к родительской Surface, в то время как дочерняя передается в качестве аргумента. Также в метод надо передать координаты размещения верхнего левого угла дочерней поверхности в координатной системе родительской. Например:

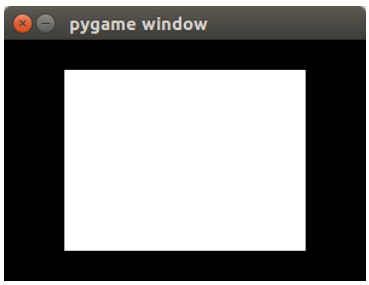
```
sc.blit(surf, (50, 20))
```

Здесь sc – основная поверхность. К ней применяется метод blit(), который на sc в ее координате 50x20 прорисовывает поверхность surf.

Пример полного кода:

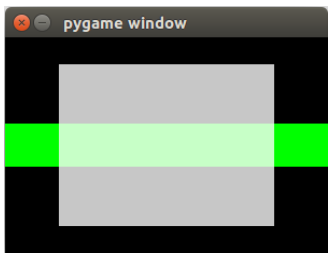
```
import pygame as pg
import sys

sc = pg.display.set_mode((300, 200))
surf = pg.Surface((200, 150))
surf.fill((255, 255, 255))
sc.blit(surf, (50, 25))
pg.display.update()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
    pg.time.delay(100)
```

Результат:



Поверхности можно делать прозрачными с помощью их метода set_alpha(). Аргумент меняется от 0 (полная прозрачность) до 255 (полная непрозрачность).

```
import pygame as pg
import sys

sc = pg.display.set_mode((300, 200))
surf = pg.Surface((200, 150))
surf.fill((255, 255, 255))
surf.set_alpha(200)

# сначала на главной поверхности
# рисуется зеленый прямоуг
pg.draw.rect(sc, (0, 255, 0),
             (0, 80, 300, 40))

# поверх накладываем полупрозрачную
# белую поверхность
sc.blit(surf, (50, 25))

pg.display.update()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
    pg.time.delay(100)
```

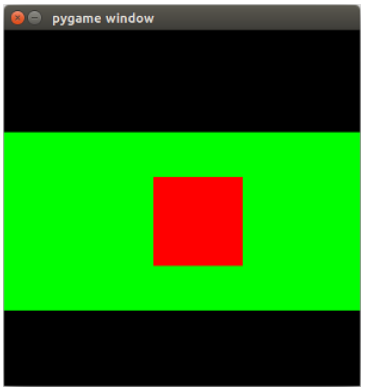


Если бы на surf располагались графические объекты, то они также стали бы полупрозрачными.

Кроме blit() и set_alpha() у поверхностей есть множество других методов. Некоторые из них будут упомянуты позже.

Если не принимать во внимание функции модуля pygame.draw, то все, что рисуется на поверхностях, делается с помощью метода blit().

Чтобы проиллюстрировать, что поверхности – это своего рода слои, запрограммируем анимацию одной поверхности (красной) на фоне другой (зеленой). Последняя может смещаться по оси *y* при клике мышью. При этом красный квадрат всегда будет двигаться ровно по центру по горизонтали зеленой поверхности, но не оконной.



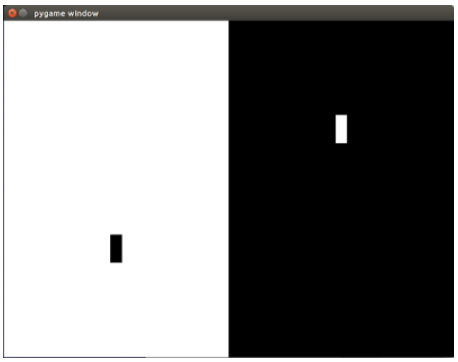

```
from random import randint
import pygame as pg
import sys

sc = pg.display.set_mode((400, 400))

background = pg.Surface((400, 200))
background.fill((0, 255, 0))
xb = 0
yb = 100

hero = pg.Surface((100, 100))
hero.fill((255, 0, 0))
x = 0
y = 50

# порядок прорисовки важен!
background.blit(hero, (x, y))
sc.blit(background, (xb, yb))

pg.display.update()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
        elif i.type == pg.MOUSEBUTTONUP:
            yb = randint(0, 200)

    if x < 400:
        x += 2
    else:
        x = 0
```

```
    sc.fill((0, 0, 0))
    background.fill((0, 255, 0))

    background.blit(hero, (x, y))
    sc.blit(background, (xb, yb))

    pg.display.update()

    pg.time.delay(30)
```

Обратите внимание на комментарий. Сначала hero прорисовывается на background. Потом background прорисовывается на sc. Если сделать наоборот, т. е. две строчки кода поменять местами, то при обновлении окна красного квадрата вы не увидите. Потому что в этом случае на sc отрисуется "старая версия" background, когда на нем еще не было hero.

Также отметим последовательность прорисовок в главном цикле игры. Сначала заливаются оба фона, иначе на них останется "след" от предыдущей итерации цикла. Далее надо заново наложить на каждый слой дочернюю для него поверхность. После этого все окно обновляется функцией update().

Рассмотрим более сложный пример. Напишем программу, в которой окно условно разделено на две половины. Если пользователь кликает по его левой части, то здесь запускается анимация. Если кликает по правой, то активность появляется здесь, при этом анимация на другой половине должна останавливаться. Пусть действием будет "взлет ракеты".



Поскольку похожих объектов будет как минимум два, то уместно написать свой класс, от которого создавать эти объекты.

```
import pygame
import sys

WIN_WIDTH = 800
WIN_HEIGHT = 600
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)


class Rocket:
    # ширина и высота у всех
    # экземпляров-ракет будут одинаковы
    width_rocket = 20
    height_rocket = 50
```

```
    def __init__(self, surface, color):
        """Конструктору необходимо передать
        поверхность, по которой будет летать
        ракета и цвет самой ракеты"""
        self.surf = surface
        self.color = color
        # Методы поверхности get_width() и
        # get_height() возвращают ее размеры.
        # Координаты верхнего левого угла
        # ракеты устанавливаются так,
        # чтобы ракета летела ровно по центру
        # поверхности по горизонтали
        # и появлялась снизу.
        self.x = surface.get_width()//2 \
                 - Rocket.width_rocket//2
        self.y = surface.get_height()

    def fly(self):
        """Вызов метода fly() поднимает
        ракету на 3 пикселя.
        Если ракета скрывается вверху,
        она снова появится снизу"""
        pygame.draw.rect(
            self.surf, self.color,
            (self.x, self.y,
             Rocket.width_rocket,
             Rocket.height_rocket))
        self.y -= 3
        # Если координата y ракеты уходит за
        # -50, то значит она
        # полностью скрылась вверху.
        if self.y < -Rocket.height_rocket:
            # Поэтому перебрасываем ракету
            # под нижнюю границу окна.
            self.y = WIN_HEIGHT


sc = pygame.display.set_mode(
    (WIN_WIDTH, WIN_HEIGHT))

# левая белая поверхность,
# равная половине окна
surf_left = pygame.Surface(
    (WIN_WIDTH//2, WIN_HEIGHT))
surf_left.fill(WHITE)

# правая черная поверхность,
# равная другой половине окна
surf_right = pygame.Surface(
    (WIN_WIDTH//2, WIN_HEIGHT))
```

```
# размещаем поверхности на главной,
# указывая координаты
# их верхних левых углов
sc.blit(surf_left, (0, 0))
sc.blit(surf_right, (WIN_WIDTH//2, 0))

# создаем черную ракету для левой
# поверхности и белую - для правой
rocket_left = Rocket(surf_left, BLACK)
rocket_right = Rocket(surf_right, WHITE)

# какая половина активна,
# до первого клика - никакая
active_left = False
active_right = False

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.MOUSEBUTTONUP:
            # если координата X клика меньше
            # половины окна, т. е. клик
            # произошел в левой половине ...
            if i.pos[0] < WIN_WIDTH//2:
                # то активируем левую,
                # отключаем правую
                active_left = True
                active_right = False
            elif i.pos[0] > WIN_WIDTH//2:
                # иначе - наоборот
                active_right = True
                active_left = False

    if active_left:
        # Если активна левая поверхность,
        # то заливаем только ее цветом,
        surf_left.fill(WHITE)
        # поднимаем ракету,
        rocket_left.fly()
        # заново отрисовываем левую
        # поверхность на главной.
        sc.blit(surf_left, (0, 0))
    elif active_right:
        surf_right.fill(BLACK)
        rocket_right.fly()
        sc.blit(surf_right, (WIN_WIDTH//2, 0))

    pygame.display.update()
    pygame.time.delay(20)
```

Заметим, что когда вызывается draw.rect() в качестве первого аргумента передается не главная оконная поверхность, а та, которая принадлежит ракете.

Практическая работа

Напишите код анимационного движения экземпляра Surface, на котором размещены несколько геометрических примитивов, нарисованных функциями модуля draw(). Этим примером иллюстрируется группировка графических объектов.

Урок 7. Класс Rect

Еще одни ключевым классом в Pygame является Rect. Его экземпляры представляют собой прямоугольные области. Они не имеют графического представления в окне игры. Ценность класса заключается в свойствах и методах, позволяющих управлять размещением поверхностей, выполнять проверку их перекрытия и др.

Rect'ы можно передавать в функцию pygame.display.update(). В этом случае будут обновляться только соответствующие им области.

Экземпляры Rect создаются не только напрямую от класса. Однако начнем с этого варианта.

```
import pygame as pg
import sys

sc = pg.display.set_mode((400, 400))

rect1 = pg.Rect((0, 0, 30, 30))
rect2 = pg.Rect((30, 30, 30, 30))

print(rect1.bottomright)  # (30, 30)
print(rect2.bottomright)  # (60, 60)

rect2.move_ip(10, 10)
print(rect2.topleft)  # (40, 40)

rect1.union_ip(rect2)
print(rect1.width)  # 70

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
```

В конструктор класса Rect передаются четыре числа – координаты *x* и *y*, ширина и высота. Мы создаем два квадрата со сторонами в 30 пикселей. Верхний левый угол первого находится в точке (0, 0), второго – (30, 30).

У объектов Rect есть более десятка свойств, связанных с их координатами и размерами. Свойство bottomright одно из них, в нем хранится координата нижнего правого угла. Понятно, что если второй квадрат начинается в точке (30, 30) и его сторона равна 30, то нижний правый угол будет в точке (60, 60).

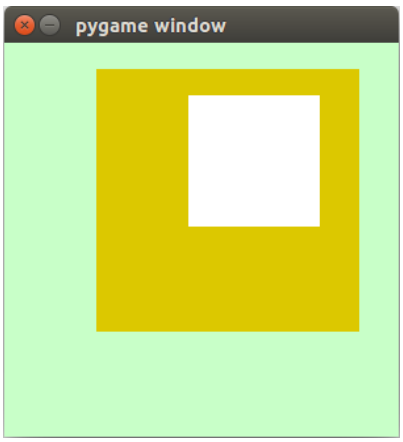
Кроме свойств, у объектов Rect есть множество методов. Метод move_ip() смещает прямоугольную область по оси *x* (первый аргумент) и *y* (второй аргумент) на указанное количество пикселей. В данном случае если второй прямоугольник смещается на 10 пикселей по обоим осям, то его левый верхний угол окажется в точке (40, 40).

Метод union_ip() присоединяет к тому прямоугольнику, к которому применяется, другой – который передается аргументом. Когда мы отодвинули второй прямоугольник на 10 пикселей, то область, заключающая в себе оба, уже будет шириной 70 пикселей, а не 60.

Методы, у которых есть суффикс _ip, изменяют тот экземпляр Rect, к которому применяются. Есть аналогичные методы без _ip (например, move(), union()), которые возвращают новый экземпляр, т. е. старый остается без изменений.

В метод blit() можно передавать не координаты места размещения Surface, а экземпляр Rect. Метод blit() сам возьмет из Rect координаты его верхнего левого угла:

```
import pygame as pg
import sys

sc = pg.display.set_mode((300, 300))
sc.fill((200, 255, 200))

surf1 = pg.Surface((200, 200))
surf1.fill((220, 200, 0))  # желтая
surf2 = pg.Surface((100, 100))
surf2.fill((255, 255, 255))  # белая

rect = pg.Rect((70, 20, 0, 0))

surf1.blit(surf2, rect)
sc.blit(surf1, rect)

pg.display.update()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
```



Мы размещаем желтую поверхность на зеленой, а белую – на желтой. В обоих случаях – в координатах (70, 20). Однако в каждом случае точка берется относительно своей родительской поверхности.

Еще один момент, на который надо обратить внимание. Прямоугольная область была определена нулевой размерностью. При этом поверхности отобразились соответственно

своим собственным размерам. Это значит, что поверхности не располагаются внутри rect'ов. Они к ним никакого отношения не имеют. Из прямоугольников blit() взял только координаты.

С другой стороны, экземпляры Rect предназначены для хранения не только координат, но и размеров поверхностей. Размеры в основном нужны для проверки коллизий. В большинстве случаев сначала создается поверхность. Далее с нее снимается "маска", т. е. создается экземпляр Rect, который будет иметь те же размеры, что и она. Все дальнейшее "взаимодействие" поверхности с другими объектами (размещение, проверка столкновений и вхождений) происходит через "связанный" с ней Rect.

```
import pygame
import sys

sc = pygame.display.set_mode((300, 300))
sc.fill((200, 255, 200))

surf2 = pygame.Surface((100, 100))
surf2.fill((255, 255, 255))  # белая

rect = surf2.get_rect()  # создается Rect

print(surf2.get_width())  # вывод 100
print(rect.width)  # 100
print(rect.x, rect.y)  # 0 0

sc.blit(surf2, rect)
pygame.display.update()

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()

    rect.x += 1

    sc.fill((200, 255, 200))
    sc.blit(surf2, rect)
    pygame.display.update()

    pygame.time.delay(20)
```

Метод поверхности get_rect() возвращает экземпляр Rect, ширина и высота которого совпадают с таковыми поверхности. В примере метод get_width() возвращает ширину поверхности, также выводится ширина прямоугольника (rect.width), чтобы показать, что они равны.

Если в get_rect() не передавать аргументы, то верхний левый угол экземпляра Rect будет в точке (0, 0).

В цикле мы изменяем координату *x* прямоугольной области, после чего передаем уже измененный rect в метод blit(). В результате поверхность будет двигаться.

Мораль такова. Нам не нужно вводить множество переменных для хранения координат и размеров. Для каждой поверхности заводится свой rect, который хранит в себе множество свойств и включает ряд полезных методов.

В get_rect() можно передавать именованные аргументы, являющиеся свойствами Rect, и устанавливать им значения. Например, surf.get_rect(topleft=(100, 50)) вернет прямоугольник, чей левый угол будет в точке (100, 50), а размер совпадать с размерами surf. Выражение surf.get_rect(centerx=100) вернет прямоугольник, координата *x* центра которого будет иметь значение 100. При этом остальные координаты будут вычислены, исходя из размеров поверхности.

Перепишем программу с двумя ракетами из предыдущего урока, используя экземпляры Rect.

```
import pygame
import sys

WIN_WIDTH = 800
WIN_HEIGHT = 600
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)


class Rocket:
    width_rocket = 20
    height_rocket = 50

    def __init__(self, surface, color):
        self.surf = surface
        self.color = color
        self.x = surface.get_width() // 2 - \
                 Rocket.width_rocket // 2
        self.y = surface.get_height()

    def fly(self):
        pygame.draw.rect(
            self.surf, self.color, (
                self.x, self.y,
                Rocket.width_rocket,
                Rocket.height_rocket))
        self.y -= 3
        if self.y < -Rocket.height_rocket:
            self.y = WIN_HEIGHT


sc = pygame.display.set_mode(
    (WIN_WIDTH, WIN_HEIGHT))

rect_left = pygame.Rect(
    (0, 0), (WIN_WIDTH // 2, WIN_HEIGHT))
rect_right = pygame.Rect(
    (WIN_WIDTH // 2, 0),
```

```
    (WIN_WIDTH // 2, WIN_HEIGHT))

surf_left = pygame.Surface(
    (rect_left.width, rect_left.height))
surf_left.fill(WHITE)

surf_right = pygame.Surface(
    (rect_right.width, rect_right.height))

sc.blit(surf_left, rect_left)
sc.blit(surf_right, rect_right)

rocket_left = Rocket(surf_left, BLACK)
rocket_right = Rocket(surf_right, WHITE)

pygame.display.update()

active_left = False
active_right = False

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.MOUSEBUTTONUP:
            if rect_left.collidepoint(i.pos):
                active_left = True
                active_right = False
            elif rect_right.collidepoint(i.pos):
                active_right = True
                active_left = False

    if active_left:
        surf_left.fill(WHITE)
        rocket_left.fly()
        sc.blit(surf_left, rect_left)
        pygame.display.update(rect_left)
    elif active_right:
        surf_right.fill(BLACK)
        rocket_right.fly()
        sc.blit(surf_right, rect_right)
        pygame.display.update(rect_right)

    pygame.time.delay(20)
```

Главное, на что здесь следует обратить внимание, – вызов функции pygame.display.update() с аргументом-прямоугольником. Таким образом, на каждой итерации главного цикла while в памяти компьютера "перерисовывается" только часть окна, что экономит его ресурсы.

```
rect_left = pygame.Rect(
    (0, 0), (WIN_WIDTH // 2, WIN_HEIGHT))
rect_right = pygame.Rect(
```

```
    (WIN_WIDTH // 2, 0),
    (WIN_WIDTH // 2, WIN_HEIGHT))
```

Создаются два экземпляра Rect. Левый начинается в верхнем левом углу окна, правый – от цента по оси x, вверху по оси y.

```
surf_left = pygame.Surface(
    (rect_left.width, rect_left.height))
surf_left.fill(WHITE)

surf_right = pygame.Surface(
    (rect_right.width, rect_right.height))
```

При создании экземпляров Surface мы указываем такую же ширину и высоту как у соответствующих им прямоугольников.

```
sc.blit(surf_left, rect_left)
sc.blit(surf_right, rect_right)
```

Левая и правая поверхности прорисовываются на главном окне. Координаты берутся из соответствующих экземпляров Rect.

```
rocket_left = Rocket(surf_left, BLACK)
rocket_right = Rocket(surf_right, WHITE)
```

Создаются два экземпляра нашего самописного класса Rocket. Конструктору надо передать поверхность и цвет.

```
active_left = False
active_right = False
```

Переменные определяют, какую анимацию проигрывать. Пока не будет произведено кликов, то никакую.

```
elif i.type == pygame.MOUSEBUTTONUP:
    if rect_left.collidepoint(i.pos):
        active_left = True
        active_right = False
    elif rect_right.collidepoint(i.pos):
        active_right = True
        active_left = False
```

Метод collidepoint() объекта Rect проверяет, находится ли точка, координаты которой были переданы в качестве аргумента, в пределах прямоугольника, к которому применяется метод. Здесь точкой являются координаты клика мыши. Если клик происходит в левом прямоугольнике, то в True устанавливается одна переменная, если в правом – то другая.

```
if active_left:
    surf_left.fill(WHITE)
    rocket_left.fly()
    sc.blit(surf_left, rect_left)
    pygame.display.update(rect_left)
elif active_right:
```

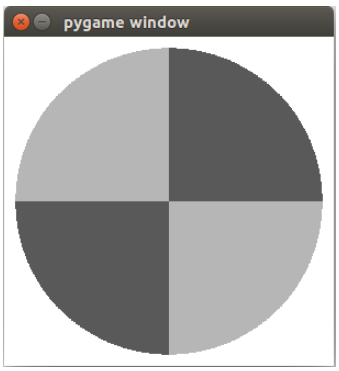
```
    surf_right.fill(BLACK)
    rocket_right.fly()
    sc.blit(surf_right, rect_right)
    pygame.display.update(rect_right)
```

В зависимости от того, какая переменная в статусе True, изменения происходят только на одной из двух дочерних поверхностей. Также как update() метод blit() вызывается на каждой итерации, иначе изменения не будут видны.

Практическая работа

Напишите программу по следующему описанию. В центре окна находится круг, изменяющий свой цвет на каждой итерации цикла. Окно условно поделено на четверти: верхнюю левую, верхнюю правую, нижнюю левую, нижнюю правую. Если нажимается клавиша 1, то обновляются только две четверти по диагонали. Если 2 – то только две другие. Нажатие нуля возобновляет обновление всей поверхность.

Примечание. Функция pygame.display.update() может принимать не только один экземпляр Rect, но и список таковых.



Урок 8. Модуль pygame.font

Классы Font и SysFont находятся в модуле pygame.font и предназначены для работы со шрифтами и текстом. Чтобы создавать от этих классов объекты, модуль pygame.font необходимо предварительно инициализировать командой pygame.font.init(), или выполнить инициализацию всех вложенных модулей библиотеки Pygame командой pygame.init().

От классов pygame.font.Font и pygame.font.SysFont создаются объекты-шрифты. Второй класс берет системные шрифты, поэтому конструктору достаточно передать имя шрифта. Конструктору Font надо передавать имя файла шрифта. Например:

```
pygame.font.SysFont('arial', 36)
pygame.font.Font('/адрес/Arial.ttf', 36)
```

Пример полного адреса в системе Linux – "/usr/share/fonts/truetype/msttcorefonts/Arial.ttf".

Второй аргумент функций – это размер шрифта в пикселях.

Узнать, какие шрифты есть в системе, можно с помощью функции get_fonts():

```
>>> pygame.font.get_fonts()
['cmmi10', 'umeminchos3', 'kacstbook', 'freesans', 'lohitpunjabi', ...
```

Узнать адрес конкретного шрифта:

```
>>> pygame.font.match_font('verdana')
'/usr/share/fonts/truetype/msttcorefonts/Verdana.ttf'
```

Вы можете скопировать шрифт в каталог программы и обращаться к нему без адреса:

```
pygame.font.Font('Verdana.ttf', 24)
```

В pygame есть шрифт по-умолчанию. Чтобы использовать его, вместо имени файла в конструктор надо передать объект None:

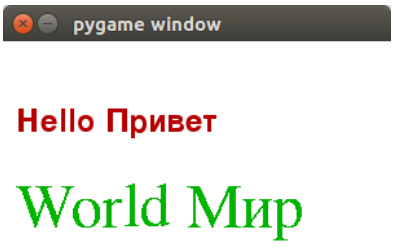
```
pygame.font.Font(None, 24)
```

От обоих классов (Font и SysFont) создаются объекты типа Font.

Метод render() экземпляра Font создает поверхность (экземпляр Surface), на которой "написан" переданный в качестве аргумента текст, шрифтом, к которому применяется метод. Вторым аргументом указывается сглаживание, третьим – цвет текста. При необходимости четвертым аргументом можно указать цвет фона.

```
import pygame
import sys
pygame.font.init()

sc = pygame.display.set_mode((300, 200))
sc.fill((255, 255, 255))

f1 = pygame.font.Font(None, 36)
text1 = f1.render('Hello Привет', True,
```

```
                  (180, 0, 0))

f2 = pygame.font.SysFont('serif', 48)
text2 = f2.render("World Мир", False,
                  (0, 180, 0))

sc.blit(text1, (10, 50))
sc.blit(text2, (10, 100))
pygame.display.update()

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
```



Рассмотрим такой пример:

```
import pygame as pg
import sys
pg.init()

sc = pg.display.set_mode((400, 300))
sc.fill((200, 255, 200))

font = pg.font.Font(None, 72)
text = font.render(
    "Hello Wold", True, (0, 100, 0))
place = text.get_rect(
    center=(200, 150))
sc.blit(text, place)

pg.display.update()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()

    pressed = pg.key.get_pressed()
    if pressed[pg.K_LEFT]:
        place.x -= 1
    elif pressed[pg.K_RIGHT]:
        place.x += 1
```

```
    sc.fill((200, 255, 200))
    sc.blit(text, place)

    pg.display.update()

    pg.time.delay(20)
```

Вспомним, что метод get_rect() экземпляра Surface возвращает объект типа Rect, чьи размеры соответствуют размерам поверхности.

Поскольку у самой поверхности нет собственных свойств-координат на родительском окне, а у Rect они есть, то по умолчанию, если get_rect() применяется без аргументов, для его верхнего левого угла устанавливаются координаты (0, 0).

В нашем примере мы передаем в get_rect() значение для свойства center порождаемой прямоугольной области. Это свойство определяет координаты центра экземпляра Rect (то, что это еще и центр главного окна, неважно). При этом остальные координаты, в том числе координаты верхнего левого угла, вычисляются автоматически, исходя из установленного центра и размеров поверхности.

Поэтому, когда вызывается метод blit(), в который в качестве второго аргумента передается созданный экземпляр Rect, то из последнего берутся координаты верхнего левого угла. Но они уже не (0, 0), а имеют значения, которые равны свойству centerx минус половина ширины и centery минус половина высоты прямоугольной области или соответствующей ей поверхности.

При зажиме стрелок на клавиатуре координата *x* прямоугольника меняется. В результате метод blit() рисует поверхность в новых координатах.

Практическая работа

У объектов Rect есть метод contains(), который проверяет, заключает ли в себе одна область (к которой применяется метод) другую (которая передается в качестве аргумента).

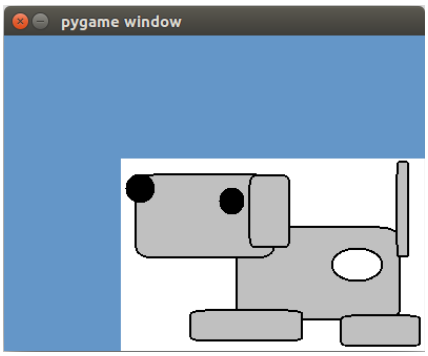
Напишите программу, в которой, если одна поверхность попадает в пределы другой, то на главной поверхности появляется какая-либо надпись. "Подвижный" экземпляр Surface должен переноситься с помощью мыши.

Урок 9. Модули pygame.image и pygame.transform

Загрузка и сохранение изображений в Pygame

Функция load() модуля pygame.image загружает изображение и создает экземпляр Surface, на котором отображено это изображение. В load() передается имя файла. "Родным" форматом является BMP, однако если функция pygame.image.get_extended() возвращает истину, то можно загружать ряд других форматов: PNG, GIF, JPG и др.

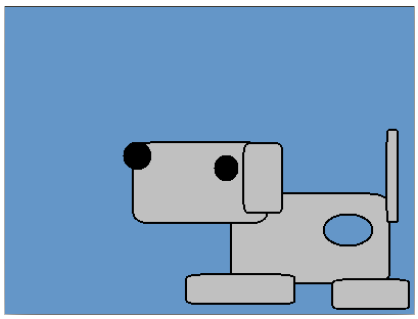
```
import pygame as pg
import sys

W = 400
H = 300

sc = pg.display.set_mode((W, H))
sc.fill((100, 150, 200))

dog_surf = pg.image.load('dog.bmp')
dog_rect = dog_surf.get_rect(
    bottomright=(W, H))
sc.blit(dog_surf, dog_rect)

pg.display.update()

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()

    pg.time.delay(20)
```



Если у изображения нет прозрачного слоя, но он необходим, то следует воспользоваться методом set_colorkey() класса Surface:

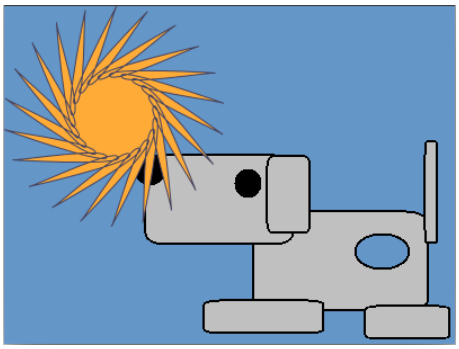
```
dog_surf = pg.image.load('dog.bmp')
dog_surf.set_colorkey((255, 255, 255))
```

Все пиксели, цвет которых совпадает с переданным в set_colorkey() значением, станут прозрачными.



У формата PNG с альфа-каналом (когда для точек можно настраивать степень прозрачности; обычно устанавливается полностью прозрачный фон) таких проблем нет:

```
sun_surf = pg.image.load('sun.png')
sun_rect = sun_surf.get_rect()
sc.blit(sun_surf, sun_rect)
```



Ко всем экземплярам Surface рекомендуется применять метод convert(), который, если не передавать аргументы, переводит формат кодирования пикселей поверхности в формат кодирования пикселей главной поверхности. При выполнении игры это ускоряет отрисовку поверхностей.

Если поверхность была создана на базе изображения с альфа-каналом, то вместо convert() надо использовать метод convert_alpha(), так как первый удаляет прозрачные пиксели (вместо них будет черный цвет). Таким образом, код загрузки и обработки изображений разных форматов должен выглядеть примерно так:

```
dog_surf = pg.image.load(
    'dog.bmp').convert()

sun_surf = pg.image.load(
    'sun.png').convert_alpha()
```

Что по смыслу равносильно:

```
...
dog_surf = pg.image.load('dog.bmp')
dog_surf = dog_surf.convert()
...
```

Метод convert() возвращает новую, конвертированную, поверхность. Он не изменяет ту, к которой применяется.

В модуле pygame.image есть функция save(), которая позволяет сохранять переданную ей поверхность (не обязательно главную) в формат BMP, TGA, PNG, JPEG. Пример:

```
while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYUP \
                and i.key == pygame.K_s:
            pygame.image.save(sc, 'day.png')

    pygame.time.delay(20)
```

Изменение поверхностей

Модуль pygame.transform содержит функции для изменения поверхностей. Некоторые трансформации (например, изменение размера) приводят к ухудшению изображения из-за потери части пикселей. В таких случаях надо сохранять исходную поверхность и выполнять трансформации от нее.

Функции модуля transform, которые изменяют поверхности, возвращают новые. Первым аргументом им передается исходный Surface. Ниже приведены примеры использования наиболее востребованных функций.

Функция flip() переворачивает Surface по горизонтали и вертикали, к потери качества не приводит. Указывается поверхность и булевыми значениями оси переворота.

```
import pygame
import sys

sc = pygame.display.set_mode((400, 300))
sc.fill((100, 150, 200))

dog_surf = pygame.image.load(
    'dog.bmp').convert()
dog_surf.set_colorkey(
    (255, 255, 255))
dog_rect = dog_surf.get_rect(
    center=(200, 150))
sc.blit(dog_surf, dog_rect)

pygame.display.update()

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYUP \
                and i.key == pygame.K_f:
            # собака перевернется
            # слева направо
```

```
            flip = pygame.transform.flip(
                dog_surf, True, False)
            sc.fill((100, 150, 200))
            sc.blit(flip, dog_rect)
            pygame.display.update(dog_rect)

    pygame.time.delay(20)
```

Поворот и изменение размера:

```
import pygame
import sys

sc = pygame.display.set_mode((400, 300))
sc.fill((100, 150, 200))

dog_surf = pygame.image.load(
    'dog.bmp').convert()
dog_surf.set_colorkey(
    (255, 255, 255))
dog_rect = dog_surf.get_rect(
    center=(200, 150))
sc.blit(dog_surf, dog_rect)
pygame.display.update()

# ждем 1 секунду перед изменением
pygame.time.wait(1000)

sc.fill((100, 150, 200))
# уменьшаем в два раза
scale = pygame.transform.scale(
    dog_surf, (dog_surf.get_width() // 2,
               dog_surf.get_height() // 2))

scale_rect = scale.get_rect(
    center=(200, 150))

sc.blit(scale, scale_rect)

pygame.display.update(dog_rect)
pygame.time.wait(1000)

sc.fill((100, 150, 200))
# поворачиваем на 45 градусов
rot = pygame.transform.rotate(
    dog_surf, 45)
rot_rect = rot.get_rect(
    center=(200, 150))
sc.blit(rot, rot_rect)
pygame.display.update()

while 1:
```

```
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
    pygame.time.delay(20)
```

Практическая работа

Допустим, у вас есть такое изображение вида сверху машины:

Напишите программу управления ее движением с помощью стрелок клавиатуры (вверх, вниз, влево, вправо) так, чтобы объект всегда двигался головой вперед.

Урок 10. Классы Sprite и Group

В программировании игр спрайтом называют объект, который предстает перед пользователем в виде анимированного изображения и в большинстве случаев предполагает взаимодействие с ним. Другими словами, все что в игре не является фоном, а интерактивным объектом-картинкой – это спрайт.

Хотя каждый спрайт может быть уникальным, у всех есть нечто общее, что в pygame вынесено в отдельный класс Sprite, находящийся в модуле pygame.sprite.

На базе этого класса следует создавать собственные классы спрайтов и уже от них объекты. Таким образом, класс pygame.sprite.Sprite играет роль своего рода абстрактного класса. Хотя таковым не является, можно создавать объекты непосредственно от Sprite.

В модуле pygame.sprite кроме класса Sprite есть класс Group и родственные ему, которые предназначены для объединения спрайтов в группы. Это позволяет вызывать один метод группы, который, например, обновит состояние всех спрайтов, входящих в эту группу.

Почти все предопределенные методы класса pygame.sprite.Sprite касаются добавления экземпляра в группу, удаления из нее, проверки вхождения. Только метод update() затрагивает поведение самого спрайта, этот метод следует переопределить в производном от Sprite классе.

Рассмотрим, как это работает. В примерах кода ниже сначала одна, а потом и множество машинок перемещаются сверху вниз. Каждая такая машинка – объект-спрайт, созданный от класса Car, который является дочерним от Sprite.

В конструкторе производного от Sprite класса необходимо вызвать конструктор родительского класса, а также обзавестись экземплярами Surface и Rect, имена которых должны быть соответственно self.image и self.rect. Так надо, чтобы с экземплярами класса могли работать методы группы. В остальном вы можете добавлять любые атрибуты.

Как создается поверхность (а также прямоугольная область), неважно. В примере ниже это делается с помощью функции load(). Однако в конструктор может передаваться уже подготовленный экземпляр Surface.

```
from random import randint
import pygame as pg
import sys

W = 400
H = 400
WHITE = (255, 255, 255)


class Car(pg.sprite.Sprite):
    def __init__(self, x, filename):
        pg.sprite.Sprite.__init__(self)
        self.image = pg.image.load(
            filename).convert_alpha()
```

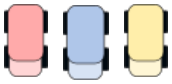
```
        self.rect = self.image.get_rect(
            center=(x, 0))


sc = pg.display.set_mode((W, H))

# координата x будет случайна
car1 = Car(randint(1, W), 'car1.png')

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()

    sc.fill(WHITE)
    sc.blit(car1.image, car1.rect)
    pg.display.update()
    pg.time.delay(20)

    # машинка ездит сверху вниз
    if car1.rect.y < H:
        car1.rect.y += 2
    else:
        car1.rect.y = 0
```

В данном случае мы изменяем свойства экземпляра за пределами класса. Правильней будет делать это в методе update():

```
...
class Car(pg.sprite.Sprite):
    def __init__(self, x, filename):
        pg.sprite.Sprite.__init__(self)
        self.image = pg.image.load(
            filename).convert_alpha()
        self.rect = self.image.get_rect(
            center=(x, 0))

    def update(self):
        if self.rect.y < H:
            self.rect.y += 2
        else:
            self.rect.y = 0


sc = pg.display.set_mode((W, H))

# координата x будет случайна
car1 = Car(randint(1, W), 'car1.png')

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
```

```
            sys.exit()

    sc.fill(WHITE)
    sc.blit(car1.image, car1.rect)
    pg.display.update()
    pg.time.delay(20)

    car1.update()
```

Теперь представим, что у нас не одна машинка, а три:



```
...
car1 = Car(randint(1, W), 'car1.png')
car2 = Car(randint(1, W), 'car2.png')
car3 = Car(randint(1, W), 'car3.png')

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()

    sc.fill(WHITE)
    sc.blit(car1.image, car1.rect)
    sc.blit(car2.image, car2.rect)
    sc.blit(car3.image, car3.rect)
    pg.display.update()
    pg.time.delay(20)

    car1.update()
    car2.update()
    car3.update()
```

Если будет 100 машинок, придется 100 раз вызвать blit() и update(). Класс Group решает эту проблему. Добавлять спрайты в группу можно методом add() группы (по одной или все вместе).

У групп есть методы update() и draw(). Метод update() группы вызывает методы update() всех входящих в нее объектов. А метод draw() выполняет метод blit(). При этом в draw() надо передать поверхность, на которой будет происходить отрисовка:

```
...
cars = pg.sprite.Group()
cars.add(Car(randint(1, W), 'car1.png'),
         Car(randint(1, W), 'car2.png'))
cars.add(Car(randint(1, W), 'car3.png'))

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
```

```
    sc.fill(WHITE)

    cars.draw(sc)

    pg.display.update()
    pg.time.delay(20)

    cars.update()
```

Допустим, мы хотим, чтобы новые машинки появлялись постоянно и в разные моменты времени, двигались с разной скоростью, а выезд объекта за пределы экрана обозначал бы, что он исчезает.

Потребуется таймер, который устанавливается вызовом функции pygame.time.set_timer(). В примере ниже через каждые 3 секунды будет генерироваться событие, значение поля type которого совпадает с константой pygame.USEREVENT. И как только это событие будет происходить, будет создаваться новый объект.

```
from random import randint
import pygame as pg
import sys

pg.time.set_timer(pg.USEREVENT, 3000)

W = 400
H = 400
WHITE = (255, 255, 255)
CARS = ('car1.png', 'car2.png', 'car3.png')
# для хранения готовых машин-поверхностей
CARS_SURF = []

# надо установить видео режим
# до вызова image.load()
sc = pg.display.set_mode((W, H))

for i in range(len(CARS)):
    CARS_SURF.append(
        pg.image.load(CARS[i]).convert_alpha())


class Car(pg.sprite.Sprite):
    def __init__(self, x, surf, group):
        pg.sprite.Sprite.__init__(self)
        self.image = surf
        self.rect = self.image.get_rect(
            center=(x, 0))
        # добавляем в группу
        self.add(group)
        # у машин будет разная скорость
        self.speed = randint(1, 3)
```

```
    def update(self):
        if self.rect.y < H:
            self.rect.y += self.speed
        else:
            # теперь не перебрасываем вверх,
            # а удаляем из всех групп
            self.kill()


cars = pg.sprite.Group()

# добавляем первую машину,
# которая появляется сразу
Car(randint(1, W),
    CARS_SURF[randint(0, 2)], cars)

while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()
        elif i.type == pg.USEREVENT:
            Car(randint(1, W),
                CARS_SURF[randint(0, 2)], cars)

    sc.fill(WHITE)

    cars.draw(sc)

    pg.display.update()
    pg.time.delay(20)

    cars.update()
```

Метод kill() спрайта удаляет его из всех групп, в которых он содержится. Есть метод remove(), который удаляет только из указанных в качестве аргумента групп. У спрайтов также как у групп есть метод add(). Только в данном случае ему передается не объект, а группа.

Практическая работа

В модуле pygame.sprite есть ряд функций для проверки коллизий спрайтов. Одна из них spritecollideany() проверяет, столкнулся ли конкретный спрайт с любым из спрайтов из группы. Функция принимает первым аргументом спрайт, чья коллизия проверяется, вторым – группу.

Измените программу выше так, чтобы машинки появлялись чаще. Добавьте спрайт, который "едет" навстречу всем другим и управляется стрелками влево и вправо на клавиатуре. Цель игры – не допустить столкновения. Если оно происходит, то программа завершается.

Урок 11. Класс Sound и модуль pygame.mixer.music

В Pygame для работы с аудио предназначены модули **pygame.mixer** и **pygame.mixer.music**. Модули похожи, однако pygame.mixer в первую очередь адаптирован для добавления и настройки звуковых эффектов в игре. В то время как pygame.mixer.music – для добавления фоновой музыки.

Функция pygame.mixer.music.load() загружает потоковое аудио, т. е. не грузит файл целиком, а делает это отдельными порциями. В результате можно проигрывать только один файл за раз. Однако можно ставить файлы в очередь функцией queue(). Поддерживает в том числе формат mp3 (но не в Ubuntu).

С другой стороны, в pygame.mixer ключевым является класс Sound. Он позволяет загружать, проигрывать и выполнять ряд других действий с файлами форматов wav или ogg. При создании экземпляра Sound в конструктор передается имя файла.

В примере ниже подгружается фоновая музыка: pygame.mixer.music.load(). Функция не возвращает никакого "музыкального" объекта, поэтому результат ее вызова не присваивается переменной.

С помощью функции music.play() файл начинает проигрываться. Если требуется зациклить композицию, то в play() передается число -1. Положительный аргумент указывает на количество повторов + одно дополнительное. То есть, если надо проиграть композицию 2 раза, то в функцию передается число 1.

В программе при нажатии на клавишу 1 клавиатуры музыка ставится на паузу: music.pause(). Клавиша 2 уменьшает громкость в два раза: music.set_volum(0.5). Нажатие 3 возвращает громкость на прежний уровень. Функция unpause() вызывается на случай, если до этого музыка была выключена (клавишей 1).

В примере создаются два объекта типа Sound. У них есть свой метод play(). В данном случае файлы проигрываются при клике левой и правой кнопками мыши. Объекты Sound могут проигрываться одновременно, так как обычно принадлежат разным каналам. Если требуется более тонкое управление звуками, дополнительно используют класс Channel.

Программа ниже в процессе выполнения интерпретатором python3.6 может выбросить ошибку, в 3.8 – все нормально. На официальном сайте pygame.org рекомендуется использовать Pygame с версией Питона от 3.7.7.

```
import pygame as pg
import sys
pg.init()
sc = pg.display.set_mode((400, 300))

pg.mixer.music.load('Beethoven.ogg')
pg.mixer.music.play()

sound1 = pg.mixer.Sound('boom.wav')
sound2 = pg.mixer.Sound('one.ogg')
```

```
while 1:
    for i in pg.event.get():
        if i.type == pg.QUIT:
            sys.exit()

        elif i.type == pg.KEYUP:
            if i.key == pg.K_1:
                pg.mixer.music.pause()
                # pygame.mixer.music.stop()
            elif i.key == pg.K_2:
                pg.mixer.music.unpause()
                # pygame.mixer.music.play()
                pg.mixer.music.set_volume(0.5)
            elif i.key == pg.K_3:
                pg.mixer.music.unpause()
                # pygame.mixer.music.play()
                pg.mixer.music.set_volume(1)

        elif i.type == pg.MOUSEBUTTONUP:
            if i.button == 1:
                sound1.play()
            elif i.button == 3:
                sound2.play()

    pg.time.delay(20)
```

Если закомментировать вызовы функций pause() и unpause() и раскомментировать stop() и play(), то результат будет схож. Разница в том, что при использовании комбинации stop-play файл начнет проигрываться сначала, а при pause-unpause продолжится с места останова.

Если у вас нет файлов wav или ogg для тестов, можете найти немного в каталоге data модуля pygame.examples. Модуль находится в папке библиотеки pygame, адрес которой можно посмотреть так:

```
>>> import pygame
>>> pygame.__file__
'/home/.../pygame/__init__.py'
```

Практическая работа

Добавьте фоновую музыку и звук столкновения в игру из практической работы предыдущего урока.

Окно игры должно закрываться только после того, как звук столкновения полностью проиграется. В зависимости от решения вам может понадобиться метод get_length() объекта типа Sound. Метод возвращает продолжительность звука, выраженную в секундах (тип float).

Решения практических работ

Урок 2. Каркас игры на Pygame

В модуле pygame.display есть функция set_caption(). Ей передается строка, которую она устанавливает в качестве заголовка окна. Сделайте так, чтобы каждую секунду заголовок окна изменялся.

Функция для установки заголовка вызывается таким образом:

```
pygame.display.set_caption("То, что вы хотите видеть в заголовке окна")
```

Пример программы:

```
import pygame
import sys

titles = ['One', 'Two', 'Three', 'Four']
id_title = 0

pygame.display.set_mode((500, 100))

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()

    pygame.display.set_caption(titles[id_title])

    if id_title < len(titles)-1:
        id_title += 1
    else:
        id_title = 0

    pygame.time.delay(1000)
```

Заголовок окна меняется каждую секунду, потому что в delay() передается 1000 мс. Строки для заголовка берутся из списка. Если достигается конец списка, индекс снова приравнивается к нулю и перебор идет сначала.

Урок 3. Модуль draw – геометрические примитивы

Написать код анимации квадрата, который перемещается от левой границе к правой, касается ее, но не исчезает за ней. После чего возвращается назад – от правой границе к левой, касается ее, опять двигается вправо. Цикл повторяется до завершения программы.

```
import pygame
import sys

FPS = 60
WIN_WIDTH = 400
WIN_HEIGHT = 100

WHITE = (255, 255, 255)
ORANGE = (255, 150, 100)

RIGHT = 'to the right'
LEFT = 'to the left'

clock = pygame.time.Clock()
sc = pygame.display.set_mode((WIN_WIDTH, WIN_HEIGHT))

# размеры и координаты левого угла
side = 60  # у квадрата ширина и высота равны
x = 0
y = WIN_HEIGHT // 2 - side // 2  # выравнивание по центру по вертикали
# если не поняли с Y, просто присвойте удобную вам координату

direction = RIGHT  # направление движения

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()

    sc.fill(WHITE)
    pygame.draw.rect(sc, ORANGE, (x, y, side, side))
    pygame.display.update()

    # меняем направление, когда сталкиваемся с краями
    if x + side >= WIN_WIDTH:  # (x + side) - координата правой границы
квадрата
        direction = LEFT
    elif x <= 0:
        direction = RIGHT

    if direction == RIGHT:
        x += 3
    else:  # если направление LEFT
        x -= 3
```

```
    clock.tick(FPS)
```

Здесь приходится вводить "флаги" направления движения, так как если координата *x* не равна левой границе (или вышла за ее пределы) и *x + side* не равно правой границе (или значение вышло за пределы), то невозможно определить, в какую сторону должен двигаться объект. Без флагов каждая следующая итерация главного цикла не будет "знать", куда двигался квадрат в предыдущей. По промежуточному значению *x* этого сказать невозможно.

Урок 4. События клавиатуры

Измените приведенную в уроке программу так, чтобы круг с той же скоростью, т. е. постепенно, возвращался назад в исходную точку, когда клавиша отпускается.

```
import pygame
import sys

FPS = 60
W = 700  # ширина экрана
H = 300  # высота экрана
WHITE = (255, 255, 255)
BLUE = (0, 70, 225)
RIGHT = "to the right"
LEFT = "to the left"
STOP = "stop"

sc = pygame.display.set_mode((W, H))
clock = pygame.time.Clock()

# координаты и радиус круга
x = W // 2
y = H // 2
r = 50

motion = STOP

while 1:
    sc.fill(WHITE)
    pygame.draw.circle(sc, BLUE, (x, y), r)
    pygame.display.update()

    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYDOWN:
            if i.key == pygame.K_LEFT:
                motion = LEFT
            elif i.key == pygame.K_RIGHT:
                motion = RIGHT
        elif i.type == pygame.KEYUP:
            if i.key in [pygame.K_LEFT, pygame.K_RIGHT]:
                motion = STOP

    if motion == LEFT:
        x -= 3
    elif motion == RIGHT:
        x += 3
    elif motion == STOP and x != W//2:
        if x < W//2:
```

```
            x += 3
        else:
            x -= 3
    clock.tick(FPS)
```

Объект должен двигаться назад при соблюдении двух условий:

1. он не двигается ни в какую сторону (motion == STOP),
2. он находится не в центре окна (x != W//2).

Первое условие соблюдается, когда никакая клавиша не нажата или клавиша была отжата.

Направление возвратного движения определяется по значению координаты *x*. Если оно меньше, чем значение координаты центра окна, то объект надо двигать вправо, т. е. увеличивать его *x*. Если больше, то надо двигать влево.

Урок 5. События мыши

Напишите код в котором имитируется полет снаряда (пусть его роль сыграет круг) в место клика мышью. Снаряд должен вылетать из нижнего края окна и лететь вверх, т. е. изменяться должна только координата *y*. Пока летит один, другой не должен появляться. Когда снаряд достигает цели, должен имитировать взрыв, например, в этом месте прорисовываться квадрат.

```
import pygame
import sys

FPS = 60
W = 700  # ширина экрана
H = 400  # высота экрана
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
ORANGE = (255, 100, 50)

pygame.init()
sc = pygame.display.set_mode((W, H))
clock = pygame.time.Clock()

bomb = False  # летит ли бомба
x_goal = 0  # координаты цели
y_goal = 0
y_bomb = H  # координата бомбы

while 1:
    sc.fill(WHITE)

    pygame.display.update()

    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.MOUSEBUTTONDOWN:
            # если был клик, но другая бомба не летит
            if i.button == 1 and not bomb:
                bomb = True
                x_goal = i.pos[0]
                y_goal = i.pos[1]

    # если бомба летит, но пока не достигла цели
    if bomb and y_bomb > y_goal:
        pygame.draw.circle(sc, BLACK, (x_goal, y_bomb), 15)
        y_bomb -= 10
        pygame.display.update()
    # если бомба уже достигла цели
    elif bomb and y_bomb <= y_goal:
        pygame.draw.rect(sc, ORANGE, (x_goal - 25, y_bomb - 15, 50, 30))
```

```
        pygame.display.update()
        pygame.time.delay(1000)  # задержка, чтобы увидеть взрыв
        bomb = False
        y_bomb = H

    clock.tick(FPS)
```

Если происходит клик и при этом другой снаряд не летит (not bomb), то снимаем координаты места клика и присваиваем их координатам цели (x_goal, y_goal). Также запускаем снаряд (bomb = True).

На каждой итерации главного цикла проверяем, достиг ли снаряд, если он есть, цели. Если не достиг (y_bomb > y_goal), то прорисовываем его, после чего меняем его координату *y*.

Когда объект достигает цели, рисуется другая картинка, и снаряд обезвреживается (bomb = False). При этом координата *y* потенциальной новой бомбы снова приравнивается к низу окна.

Чтобы "взрыв" был не слишком коротким по времени, надо прорисовывать прямоугольник в нескольких итерациях. Реализация этого заметно усложнила бы код. Поэтому в данном случае просто используется задержка с помощью функции delay(). Это не совсем правильно, так как во время задержки окно зависает и не реагирует на клики.

Урок 6. Класс Surface и метод blit()

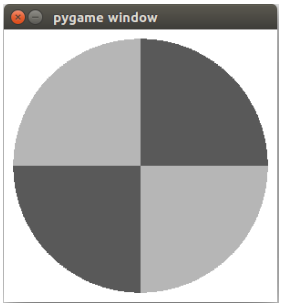
Напишите код анимационного движения экземпляра Surface, на котором размещены несколько геометрических примитивов, нарисованных функциями модуля draw(). Этим примером иллюстрируется группировка графических объектов.

```
import pygame
import sys

WIDTH = 700
HEIGHT = 300

sc = pygame.display.set_mode((WIDTH, HEIGHT))
surf = pygame.Surface((200, 200))

pygame.draw.circle(surf, (255, 0, 0), (75, 75), 75)
pygame.draw.line(surf, (0, 255, 0), (0, 150), (200, 150), 5)

x = 0
y = HEIGHT // 2 - 100

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()

    sc.fill((0, 0, 0))
    sc.blit(surf, (x, y))
    pygame.display.update()
    x += 3
    pygame.time.delay(20)
```

В функции circle() и line() передается поверхность surf, на которой группируются фигуры. Сама surf константна, в цикле на ней ничего не меняется. Поскольку анимация происходит только на главной поверхности, то заливать надо только ее. Анимация обеспечивается изменением координат, передаваемых в метод blit() главной поверхности.

Урок 7. Класс Rect

Напишите программу по следующему описанию. В центре окна находится круг, изменяющий свой цвет на каждой итерации цикла. Окно условно поделено на четверти: верхнюю левую, верхнюю правую, нижнюю левую, нижнюю правую. Если нажимается клавиша 1, то обновляются только две четверти по диагонали. Если 2 – то только две другие. Нажатие нуля возобновляет обновление всей поверхность.



Примечание. Функция pygame.display.update() может принимать не только один экземпляр Rect, но и список таковых.

```
import pygame
import sys

SIDE = 300
C = SIDE//2
WHITE = (255, 255, 255)
col = 0
col_flag = '+'
TL_BR = 'top left, bottom right'
BL_TR = 'bottom left, top right'

sc = pygame.display.set_mode((SIDE, SIDE))
sc.fill(WHITE)

r1 = pygame.Rect((0, 0, C, C))  # верхний левый
r2 = pygame.Rect((C, 0, C, C))  # верхний правый
r3 = pygame.Rect((0, C, C, C))  # нижний левый
r4 = pygame.Rect((C, C, C, C))  # нижний правый

flag = 0
while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYUP:
            if i.key == pygame.K_1:
                flag = TL_BR
            elif i.key == pygame.K_2:
                flag = BL_TR
            elif i.key == pygame.K_0:
                flag = 0
```

```
    pygame.draw.circle(sc, (col, col, col), (C, C), C - 10)

    if flag == TL_BR:
        pygame.display.update([r1, r4])
    elif flag == BL_TR:
        pygame.display.update([r2, r3])
    else:
        pygame.display.update()

    pygame.time.delay(20)

    if col_flag == '+':
        col += 3
    else:
        col -= 3

    if col == 255:
        col_flag = '-'
    elif col == 0:
        col_flag = '+'
```

Урок 8. Модуль pygame.font

У объектов Rect есть метод contains(), который проверяет, заключает ли в себе одна область (к которой применяется метод) другую (которая передается в качестве аргумента).

Напишите программу, в которой, если одна поверхность попадает в пределы другой, то на главной поверхности появляется какая-либо надпись. "Подвижный" экземпляр Surface должен переноситься с помощью мыши.

```
import pygame
import sys
pygame.init()

sc = pygame.display.set_mode((400, 300))

# Область, в которую должен попадать объект,
# располагается в нижнем правом углу и имеет размер 100x100.
surf_gate = pygame.Surface((100, 100))
surf_gate.fill((0, 255, 255))
rect_gate = surf_gate.get_rect(bottomright=(400, 300))
sc.blit(surf_gate, rect_gate)

# поверхность, которая будет перемещаться
surf = pygame.Surface((50, 50))
surf.fill((255, 255, 0))  # желтая
rect = surf.get_rect()  # появится в левом верхнем углу
sc.blit(surf, rect)

# текст, который будет отображаться
font = pygame.font.Font(None, 24)
text = font.render("Yes!!!", True, (255, 255, 255))
rect_text = text.get_rect(topleft=(10, 10))

pygame.display.update()

flag = False

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        # Если клавиша мыши зажимается
        elif i.type == pygame.MOUSEBUTTONDOWN:
            # и событие происходит внутри rect
            if rect.collidepoint(i.pos):
                flag = True  # устанавливаем флаг.
        # Если клавиша мыши отжимается,
        elif i.type == pygame.MOUSEBUTTONUP:
            flag = False  # сбрасываем флаг.
```

```
    # Флаг истина, если клавиша мыши была зажата,
    # когда курсор находился в пределах желтой поверхности,
    # и еще не была отпущена.
    if flag:
        # заливаем окно для перерисовки поверхности
        # в новых координатах
        sc.fill((0, 0, 0))
        # устанавливаем центр прямоугольной области,
        # в текущую позицию курсора
        rect.center = pygame.mouse.get_pos()
        # отрисовываем объекты и обновляем окно
        sc.blit(surf_gate, rect_gate)
        sc.blit(surf, rect)
        pygame.display.update()

    # Если один прямоугольник внутри другого,
    if rect_gate.contains(rect):
        # заливаем всю поверхность,
        sc.fill((0, 0, 0))
        # отрисовываем сообщение,
        sc.blit(text, rect_text)
        # но обновляем только ту часть окна,
        # в которой отображается текст.
        pygame.display.update(rect_text)
        # Поэтому от заливки цветные поверхности не исчезнут,
        # и их не надо перерисовывать.

    pygame.time.delay(20)
```

Урок 9. Модули pygame.image и pygame.transform

Допустим, у вас есть такое изображение вида сверху машины:

Напишите программу управления ее движением с помощью стрелок клавиатуры (вверх, вниз, влево, вправо) так, чтобы объект всегда двигался головой вперед.

```
import pygame
import sys

W = 400
H = 400
WHITE = (255, 255, 255)

sc = pygame.display.set_mode((W, H))
sc.fill(WHITE)

car = pygame.image.load('car.png').convert_alpha()
rect = car.get_rect(center=(W//2, H//2))
sc.blit(car, rect)

pygame.display.update()

car1 = car  # делаем копию, чтобы не портить исходник

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYDOWN:
            if i.key == pygame.K_LEFT:
                car1 = pygame.transform.rotate(car, 90)
            elif i.key == pygame.K_RIGHT:
                car1 = pygame.transform.rotate(car, -90)
            elif i.key == pygame.K_UP:
                car1 = car
            elif i.key == pygame.K_DOWN:
                car1 = pygame.transform.rotate(car, 180)

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        rect.centerx -= 3
    elif keys[pygame.K_RIGHT]:
        rect.centerx += 3
    elif keys[pygame.K_UP]:
        rect.centery -= 3
    elif keys[pygame.K_DOWN]:
        rect.centery += 3
```

```
    sc.fill(WHITE)
    sc.blit(car1, rect)
    pygame.display.update()

    pygame.time.delay(20)
```

В цикле обработки событий проверяется нажатие стрелок. Если событие происходит, то машина поворачивается в соответствующую сторону.

В основном теле цикла while изменяется координата *x* или *y* в зависимости от того, какая клавиша зажата.

У такого варианта решения есть глюк. Если нажать вторую кнопку до того, как отпустить первую, машина может двигаться в одну сторону, а ее "морда" будет повернута в другую. Почему? Представьте, что вы зажали стрелку влево. Машина развернулась налево. Вы не отпускаете левую стрелку, и машина едет влево. Далее, не отпуская левую стрелку, нажимаете правую.

Происходит новое событие KEYDOWN, чье свойство key равно K_RIGHT. Машину разворачивает направо. Однако, поскольку среди нажатых клавиш есть K_LEFT, срабатывает ветка `if keys[pygame.K_LEFT]`, после чего до elif поток уже не доходит. Таким образом, машина продолжает двигаться влево, хотя развернулась вправо.

Чтобы вылечить это, можно ввести флаг, выполняющий роль второго условия:

```
import pygame
import sys

W = 400
H = 400
WHITE = (255, 255, 255)

sc = pygame.display.set_mode((W, H))
sc.fill(WHITE)

car = pygame.image.load('car.png').convert_alpha()
rect = car.get_rect(center=(W//2, H//2))
sc.blit(car, rect)

pygame.display.update()

car1 = car

flag = 'up'

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.KEYDOWN:
```

```
            if i.key == pygame.K_LEFT:
                car1 = pygame.transform.rotate(car, 90)
                flag = 'left'
            elif i.key == pygame.K_RIGHT:
                car1 = pygame.transform.rotate(car, -90)
                flag = 'right'
            elif i.key == pygame.K_UP:
                car1 = car
                flag = 'up'
            elif i.key == pygame.K_DOWN:
                car1 = pygame.transform.rotate(car, 180)
                flag = 'down'

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT] and flag == 'left':
        rect.centerx -= 3
    elif keys[pygame.K_RIGHT] and flag == 'right':
        rect.centerx += 3
    elif keys[pygame.K_UP] and flag == 'up':
        rect.centery -= 3
    elif keys[pygame.K_DOWN] and flag == 'down':
        rect.centery += 3

    sc.fill(WHITE)
    sc.blit(car1, rect)
    pygame.display.update()

    pygame.time.delay(20)
```

В данном случае координата будет меняться только в том случае, если зажата стрелка и до этого произошел поворот в соответствующую ей сторону.

Но и тут не все гладко. Если нажаты две стрелки, потом отпускается та, в направлении которой ехала машина, то она не поедет в сторону другой клавиши. Надо заново сгенерировать событие, отпустив и заново нажав клавишу.

Урок 10. Классы Sprite и Group

В модуле pygame.sprite есть ряд функций для проверки коллизий спрайтов. Одна из них spritecollideany() проверяет, столкнулся ли конкретный спрайт с любым из спрайтов из группы. Функция принимает первым аргументом спрайт, чья коллизия проверяется, вторым – группу.

Измените программу выше так, чтобы машинки появлялись чаще. Добавьте спрайт, который "едет" навстречу всем другим и управляется стрелками влево и вправо на клавиатуре. Цель игры – не допустить столкновения. Если оно происходит, то программа завершается.

```
from random import randint
import pygame
import sys

pygame.time.set_timer(pygame.USEREVENT, 1000)

W = 400
H = 400
WHITE = (255, 255, 255)
CARS = ('car1.png', 'car2.png', 'car3.png')
CARS_SURF = []

sc = pygame.display.set_mode((W, H))

for i in range(len(CARS)):
    CARS_SURF.append(pygame.image.load(CARS[i]).convert_alpha())


class Car(pygame.sprite.Sprite):
    def __init__(self, x, surf, group):
        pygame.sprite.Sprite.__init__(self)
        self.image = surf
        self.rect = self.image.get_rect(center=(x, 0))
        self.add(group)
        self.speed = randint(1, 3)

    def update(self):
        if self.rect.y < H:
            self.rect.y += self.speed
        else:
            self.kill()


class UserCar(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load('car.png').convert_alpha()
        self.rect = self.image.get_rect(midbottom=(W//2, H-10))
```

```
cars = pygame.sprite.Group()

Car(randint(1, W), CARS_SURF[randint(0, 2)], cars)

user = UserCar()

while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.USEREVENT:
            Car(randint(1, W), CARS_SURF[randint(0, 2)], cars)

    sc.fill(WHITE)

    cars.draw(sc)
    cars.update()

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        user.rect.x -= 2
    elif keys[pygame.K_RIGHT]:
        user.rect.x += 2

    sc.blit(user.image, user.rect)

    pygame.display.update()
    pygame.time.delay(20)

    if pygame.sprite.spritecollideany(user, cars):
        sys.exit()
```

Урок 11. Класс Sound и модуль pygame.mixer.music

Добавьте фоновую музыку и звук столкновения в игру из практической работы предыдущего урока.

Окно игры должно закрываться только после того, как звук столкновения полностью проиграется. В зависимости от решения вам может понадобиться метод get_length() объекта типа Sound. Метод возвращает продолжительность звука, выраженную в секундах (тип float).

```
from random import randint
import pygame
import sys
pygame.init()
pygame.time.set_timer(pygame.USEREVENT, 1000)

W = 400
H = 400
WHITE = (255, 255, 255)
CARS = ('car1.png', 'car2.png', 'car3.png')
CARS_SURF = []

pygame.mixer.music.load('house_lo.ogg')
pygame.mixer.music.play(-1)

# СПЕЦЭФФЕКТ
sound = pygame.mixer.Sound('boom.wav')

sc = pygame.display.set_mode((W, H))

for i in range(len(CARS)):
    CARS_SURF.append(pygame.image.load(CARS[i]).convert_alpha())


class Car(pygame.sprite.Sprite):
    def __init__(self, x, surf, group):
        pygame.sprite.Sprite.__init__(self)
        self.image = surf
        self.rect = self.image.get_rect(center=(x, 0))
        self.add(group)
        self.speed = randint(1, 3)

    def update(self):
        if self.rect.y < H:
            self.rect.y += self.speed
        else:

            self.kill()
```

```
class UserCar(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.image.load('car.png').convert_alpha()
        self.rect = self.image.get_rect(midbottom=(W//2, H-10))


cars = pygame.sprite.Group()

Car(randint(1, W), CARS_SURF[randint(0, 2)], cars)

user = UserCar()

play = True
while play:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            sys.exit()
        elif i.type == pygame.USEREVENT:
            Car(randint(1, W), CARS_SURF[randint(0, 2)], cars)

    sc.fill(WHITE)

    cars.draw(sc)
    cars.update()

    keys = pygame.key.get_pressed()
    if keys[pygame.K_LEFT]:
        user.rect.x -= 2
    elif keys[pygame.K_RIGHT]:
        user.rect.x += 2

    sc.blit(user.image, user.rect)

    pygame.display.update()
    pygame.time.delay(20)

    if pygame.sprite.spritecollideany(user, cars):
        play = False

sound_len = sound.get_length()
pygame.mixer.music.stop()
sound.play()
pygame.time.delay((int(sound_len)+1)*1000)
```

В данном случае вместо того, чтобы завершать программу прямо в главном цикле, из него выполняется выход в основную ветку. Здесь кроме того, что звук начинает проигрываться, измеряется его продолжительность, и на эту величину выполняется задержка.