Фоксфорд

Занятие №11 “Pygame”

Данилова Анна Александровна / регалии кратко

Если видео в плохом качестве:

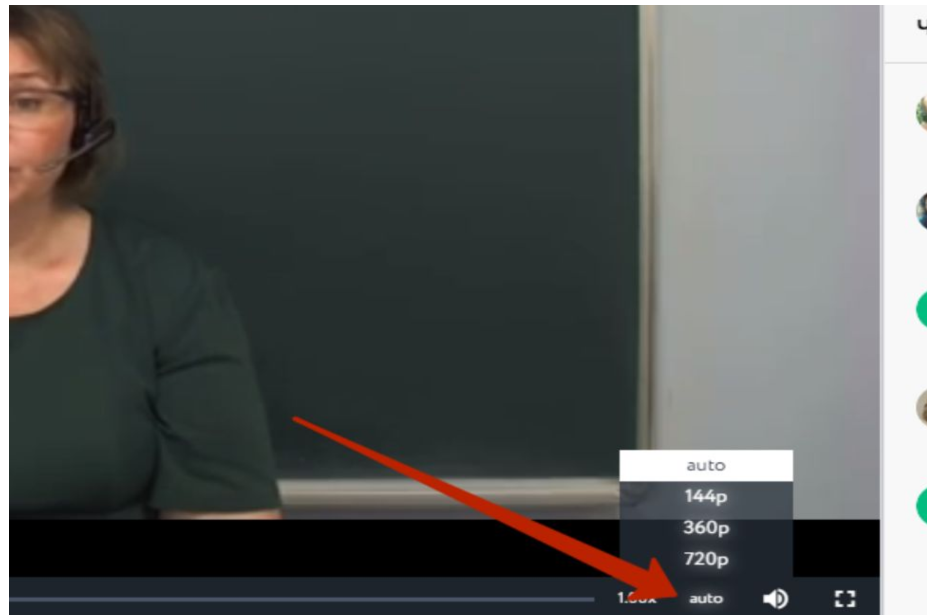
Уважаемые слушатели!

Видеозапись по умолчанию открывается в качестве "auto".

Это необходимо, чтобы видео не “зависало”. НО! Автоматический режим зависит от вашей скорости интернета. Иногда это достаточно низкое качество, неудобное для просмотра.

Вы всегда можете выбрать подходящее качество вручную!

Для этого **нажмите на кнопку, как показано на картинке:**



Данилова Анна Александровна

danilowaanna@gmail.com

https://vk.com/vasanima

Фоксфорд

План занятия

1 Объекты
2 Наследование
3 Модули

Основные блоки игры

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.

Фоксфорд

pygame

Создание окна

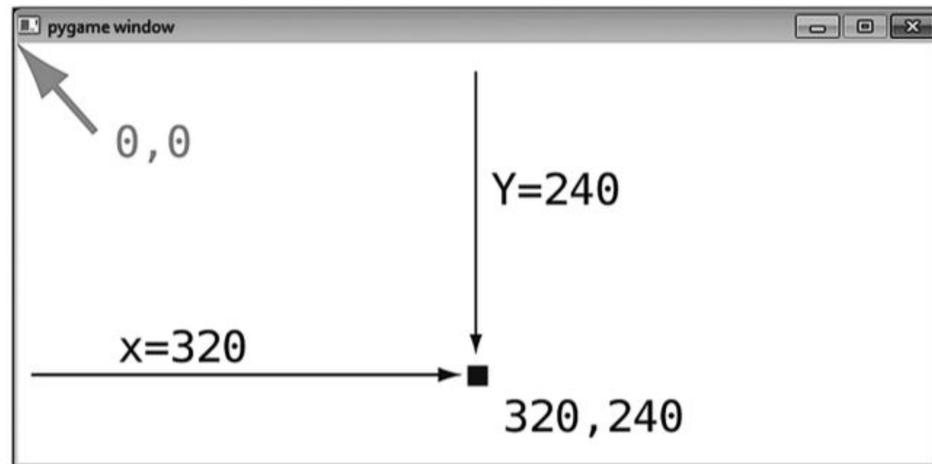
Фоксфорд

В pygame есть функция init(), которая импортирует весь инструментарий pygame, другими словами, инициализирует все модули библиотеки.

После этого можно вывести на экран главное графическое окно игры с помощью функции set_mode() модуля display, входящего в состав библиотеки pygame

Создание окна

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




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

Если выполнить этот код, то появится окно размером 600x400 пикселей и сразу закроется.

set_mode()

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.

Pygame.locals

Фоксфорд

Выражение вида pygame.RESIZABLE

обозначает обращение к той или иной константе, определенной в модуле pygame.

Часто можно встретить код, в котором перед константами не пишется имя модуля pygame.QUIT -> 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 *
```

Set_mode()

Фоксфорд

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.

update() flip()

Фоксфорд

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

update() и flip() модуля display обновляют содержимое окна игры. Это значит, что каждому пикселю заново устанавливается цвет.

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.

Создание окна

Фоксфорд

Основной цикл

Фоксфорд

Вернемся к нашим трем строчкам кода. Почему окно сразу закрывается? Очевидно потому, что программа заканчивается после выполнения этих выражений.

В tkinter для этого используется метод mainloop() экземпляра Tk().

В pygame же требуется собственноручно создать бесконечный цикл, заставляющий программу зависнуть. Итак, создадим в программе бесконечный цикл

Закрытие окна с ошибкой

Фоксфорд

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.

```
import pygame
pygame.init()
pygame.display.set_mode((600, 400))
while 1:
    for i in pygame.event.get():
        if i.type == pygame.QUIT:
            pygame.quit()
```

pygame.event.get()

Фоксфорд

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, можно записать так:

Список событий

```
...
while 1:
    # events содержит список событий
    events = pygame.event.get()
    for event in events:
        if event.type == pygame.QUIT:
            pygame.quit()
```

event.type = pygame.QUIT

Фоксфорд

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.

Завершение по исключению

Фоксфорд

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 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 необработанное, исключение.

sys.exit()

```
import pygame
import sys

pygame.init()

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

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

Данную проблему можно решить как минимум двумя способами.

Часто используют функцию exit() модуля sys.

В этом случае код выглядит примерно так

Сначала отключается pygame, потом происходит выход из программы. Такой вариант вероятно следует считать наиболее безопасным завершением.

play - завершение цикла

Фоксфорд

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

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.

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

Завершение по return

```
import pygame
pygame.init()

def main():

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

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

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

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.

Частота

Фоксфорд

Перейдем к следующему вопросу. С какой скоростью крутится цикл while?

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.

time.delay

Фоксфорд

```
import pygame

pygame.init()

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

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

    pygame.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.

time.Clock

Фоксфорд

```
import pygame

pygame.init()

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

clock = pygame.time.Clock()

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

FPS

Фоксфорд

Нередко частоту кадров выносят в отдельную константоподобную переменную:

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

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

    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.

Каркас игры

Фоксфорд

```
# здесь подключаются модули
import pygame

# здесь определяются константы, классы и функции
FPS = 60

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

# если надо до цикла отобразить объекты на экране
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:
            exit()

    # --------
    # изменение объектов и многое др.
    # --------

    # обновление экрана

     pygame.display.update()
```

pygame.draw геометрические примитивы

Фоксфорд

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

Отрисовка скелет

```
import pygame
 pygame.init()
sc = pygame.display.set_mode((300, 200))
# здесь будут рисоваться фигуры
pygame.display.update()

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

pygame.draw геометрические примитивы

Фоксфорд

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 несколько 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.

Не забываем обновлять окно

Фоксфорд

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.

pygame.draw геометрические примитивы

Фоксфорд

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.

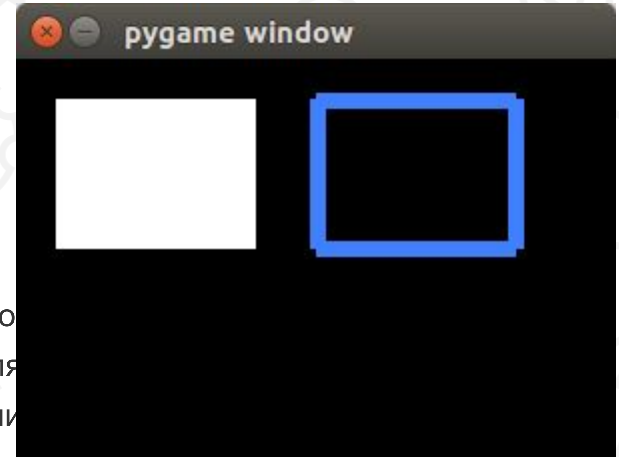
Все функции модуля draw возвращают экземпляры класса Rect – прямоугольные области, имеющие координаты, длину и ширину. Не путайте функцию rect() модуля draw и класс Rect, это разные вещи.

Прямоугольники

Фоксфорд

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 стро
незаполненным, а цвет определит цвет рамки. Третьим аргументом явля
Первые два определяют координаты верхнего левого угла прямоугольни
высоту.

pygame.draw геометрические примитивы

Фоксфорд

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.

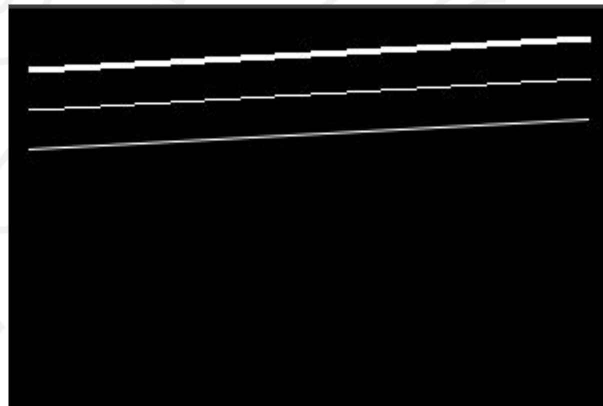
Обычно цвета выносят в отдельные переменные-константы. Это облегчает чтение кода:

```
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)
```

Линии

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])
```



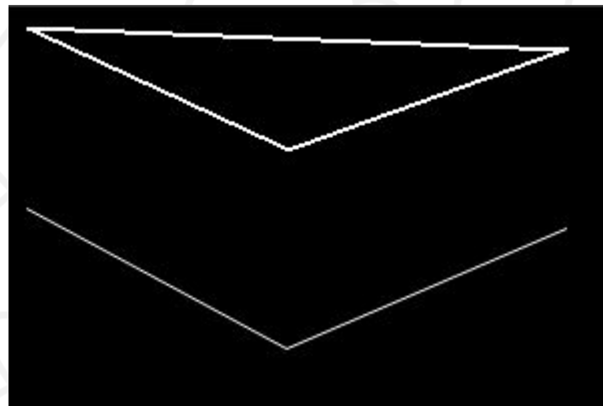
Ломаные

Фоксфорд

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]])
```

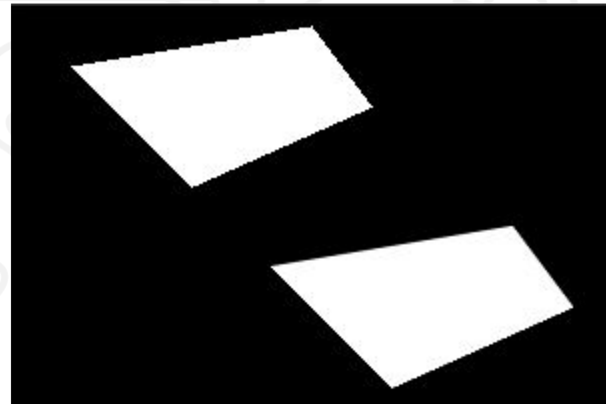


Многоугольник

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.

Круг и эллипс

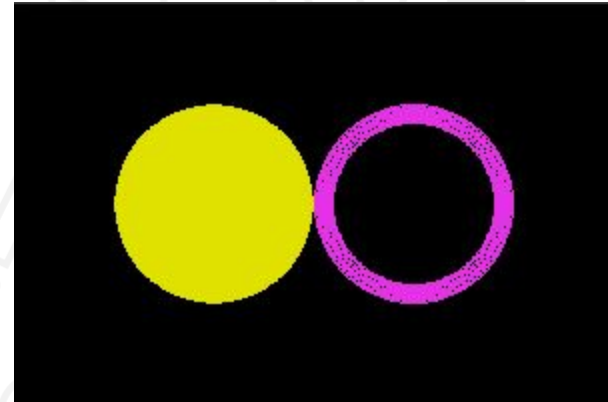
Фоксфорд

Функция circle() рисует круги.

Указывается центр окружности и радиус:

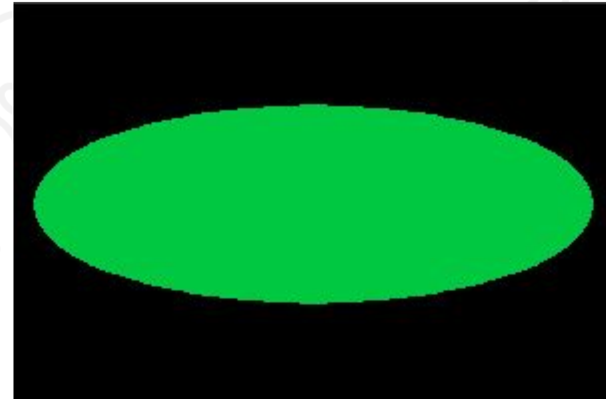
```
pygame.draw.circle(sc, YELLOW, (100, 100), 50)
```

```
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))
```



Дуги

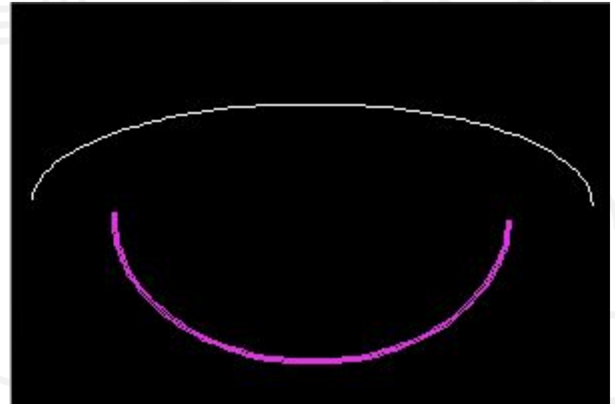
Фоксфорд

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.

ЛИСТИНГ

Анимация

Фоксфорд

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.

Анимация

Фоксфорд

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.

Код анимации круга. Объект появляется с левой стороны, доходит до правой, исчезает за ней. После этого снова появляется слева.

Листинг

События клавиатуры

Фоксфорд

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

Фоксфорд

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.

события клавиатуры

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:

listing1

события клавиатуры

Фоксфорд

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*.

события клавиатуры

Фоксфорд

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.

события клавиатуры

Фоксфорд

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.

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).

события мыши

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.

события мыши

Фоксфорд

pygame window

события мыши

Фоксфорд

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

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):

события мыши

Фоксфорд

Функцией get_pos() мы можем считывать позицию курсора, даже если он не виден. Далее в этой позиции рисуем фигуру в каждом кадре.

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.

Фоксфорд

Ваши вопросы по проекту

Идеи, как можно усовершенствовать

Фоксфорд

Спасибо за внимание!

Данилова Анна Александровна

danilowaanna@gmail.com

https://vk.com/vasanima