* [ОБЗОР КУРСА](https://lyceum.yandex.ru/courses/165/groups/1257)

[Урок PG. События](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1237)

**Игровой цикл. События**

1. [Поговорим о времени](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1237/materials/2812#1)
2. [Время в PyGame](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1237/materials/2812#2)
3. [События](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1237/materials/2812#3)
4. [События по таймеру](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1237/materials/2812#4)
5. [Холст (Surface)](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1237/materials/2812#5)
6. [Памятка по решению задач](https://lyceum.yandex.ru/courses/165/groups/1257/lessons/1237/materials/2812#6)

**Аннотация**

*В уроке поговорим об игровом цикле. Обсудим работу со временем, кадрами и событиями.*

**Поговорим о времени**

Обычно программа на Pygame, даже если она показывает статичную картинку, все равно содержит **игровой цикл**.

Главный игровой цикл — обязательный компонент любой игры. В нем происходит постоянная отрисовка игровых объектов, изменение их состояния (например, положения) и обработка событий. Прежде всего, цикл реагирует на действия пользователя.

Рассмотрим обработку завершения программы: цикл должен быть завершен по желанию пользователя.

running = True

while running:

# внутри игрового цикла ещё один цикл

# приема и обработки сообщений

for event in pygame.event.get():

# при закрытии окна

if event.type == pygame.QUIT:

running = False

# отрисовка и изменение свойств объектов

# ...

# обновление экрана

pygame.display.flip()

Игра заканчивается, когда завершается главный игровой цикл.

Если завести переменную x\_pos, занести в нее значение 0, а в цикл добавить строки:

screen.fill((0, 0, 0))

pygame.draw.circle(screen, (255, 0, 0), (x\_pos, 200), 20)

x\_pos += 1

то красный круг «поедет» вправо.

Для аккуратности лучше поместить рисование в отдельную функцию. На прошлом занятии мы называли ее draw(). Если написать в нее генерацию случайных точек, то картинка на экране будет постоянно меняться, получится эффект ряби не настроенного на канал телевизора.

**Тренировочное задание.** Реализуйте программу, моделирующую ненастроенный телевизор.

**Время в PyGame**

Не имеет большого значения, с какой скоростью мерцает телевизор из предыдущего примера. Но в играх время играет очень важную роль. На некоторых машинах движение будет идти слишком быстро, на других — слишком медленно. Это зависит как от мощности компьютера, так и от загруженности процессора.

Но разработчик игры стремится к тому, чтобы на любом компьютере движение выглядело примерно одинаково. Для этого нужно учитывать время.

*Л. Кэрролл «Алиса в стране чудес»*

В Pygame для учета времени есть специальный класс Clock в модуле time.

Нужно создать его экземпляр перед игровым циклом, а в самом цикле на каждом шаге вызывать метод tick() этого экземпляра.

Этот метод возвращает **количество миллисекунд**, прошедших с момента последнего вызова. Можно ориентироваться на него и работать с объектом игры с учетом полученного прошедшего времени.

Например, завести переменную скорости и вычислять новое положение объекта по формуле x\_pos += v \* clock.tick():

running = True

x\_pos = 0

v = 20 # пикселей в секунду

clock = pygame.time.Clock()

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

screen.fill((0, 0, 0))

pygame.draw.circle(screen, (255, 0, 0), (int(x\_pos), 200), 20)

x\_pos += v \* clock.tick() / 1000 # v \* t в секундах

pygame.display.flip()

Теперь кружок из первого примера будет перемещаться со скоростью **ровно** 20 пикселей в секунду практически равномерно.

Обратите внимание: при вычислениях x может стать нецелым, а при рисовании окружности позиция центра должна быть кортежем целых чисел. Поэтому нужно приводить x к типу int.

В простых случаях, когда особая точность не требуется, можно просто передавать в функцию tick() требуемое количество кадров (FPS — Frames per Second — кадров в секунду) и считать, что кадры рассчитываются и рисуются почти мгновенно. В этом случае метод tick() будет задерживать выполнение программы так, чтобы количество кадров было не больше переданного значения — оно будет примерно равно ему — и дальше ориентироваться на это значение:

x = 0

v = 20 # пикселей в секунду

fps = 60

clock = pygame.time.Clock()

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

screen.fill((0, 0, 0))

pygame.draw.circle(screen, (255, 0, 0), (int(x), 200), 20)

x += v / fps

clock.tick(fps)

pygame.display.flip()

Конечно, у компьютера есть предел, и 1000 fps вы не получите в любом случае. Но, на самом деле, и 30 fps вполне достаточно.

**События**

В Pygame есть модули mouse и keyboard. Они позволяют «опрашивать» мышь и клавиатуру в любой момент, то есть получать от устройств информацию. Но удобнее работать с **событиями.**

Важнее узнать, что кнопка мыши *нажалась*, чем получить информацию о том, что она *нажата*.

Любая игра также управляется событиями. Что же это за события?

Прежде всего, это события пользовательского ввода: игрок нажал клавишу на клавиатуре, подвинул мышь, нажал на кнопку закрытия окна и т. д. На каждом шаге главного игрового цикла мы разбираем накопившиеся события.

Несмотря на то, что цикл работает очень быстро, за одну итерацию наступивших событий может быть несколько. Поэтому в программе появляется второй внутренний цикл, который обрабатывает все произошедшие события (разбирает очередь событий).

Еще раз вернемся к шаблону игровой программы:

running = True

while running:

# внутри игрового цикла ещё один цикл

# приёма и обработки сообщений

for event in pygame.event.get():

# при закрытии окна

if event.type == pygame.QUIT:

running = False

# РЕАКЦИЯ НА ОСТАЛЬНЫЕ СОБЫТИЯ

# ...

# отрисовка и изменение свойств объектов

# ...

pygame.display.flip()

Обратите внимание, мы забираем события функцией get(), а не функцией wait(), как на прошлом занятии. wait() блокирует выполнение программы, пока не наступит событие. Такое поведение подходит для шахмат или пошаговых стратегий, но в **шутере** монстры не станут ждать, пока игрок выстрелит.

Таким образом, главный игровой цикл обычно выглядит примерно так:

fps = 50 # количество кадров в секунду

clock = pygame.time.Clock()

running = True

while running: # главный игровой цикл

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

# обработка остальных событий

# ...

# формирование кадра

# ...

pygame.display.flip() # смена кадра

# изменение игрового мира

# ...

# временная задержка

clock.tick(fps)

Каждое событие содержит в себе его тип и параметры. Например, события от мыши содержат позицию курсора и информацию о том, какая кнопка была нажата или отпущена.

Приведем список основных типов событий с их атрибутами:

| **event.type** | **атрибуты** |
| --- | --- |
| QUIT | нет |
| KEYDOWN | unicode, key, mod (например, shift, ctrl...) |
| KEYUP | key, mod |
| MOUSEMOTION | pos (кортеж текущих координат), rel (кортеж координат относительно предыдущего события), buttons (кортеж номеров нажатых кнопок в момент движения) |
| MOUSEBUTTONUP | pos, button |
| MOUSEBUTTONDOWN | pos, button |

Например, код:

while running:

screen.fill((0, 0, 0))

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if event.type == pygame.MOUSEMOTION:

pygame.draw.circle(screen, (0, 0, 255), event.pos, 20)

pygame.display.flip()

clock.tick(50)

отображает при движении мыши синий круг под курсором.

Обратите внимание: круг исчезает, если мышь не двигать. Почему? Как это можно исправить?

Pygame поставляется с большим количеством [примеров, небольших программ](https://www.pygame.org/docs/ref/examples.html), иллюстрирующих ее возможности. Примеры устанавливаются вместе с библиотекой в виде модуля examples.

Хорошо помогает разобраться с событиями пример eventlist. Его можно запустить из командной строки

python -m pygame.examples.eventlist

Или кодом (из среды программирования):

import pygame.examples.eventlist

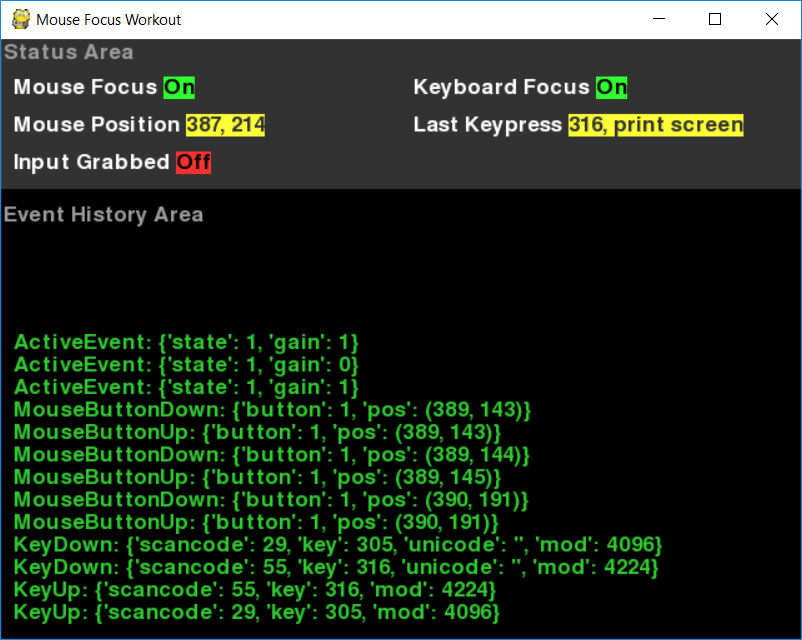
pygame.examples.eventlist.main()

А еще лучше — узнать местоположение папки с примерами с помощью следующей мини-программы:

import pygame.examples

pygame.examples.\_\_file\_\_

и скопировать оттуда в среду исходный код из файла eventlist.py. Тогда его можно будет изменять.



Поэкспериментируйте с кодом этого примера.

**Тренировочное задание.** По [документации по модулю mouse](https://www.pygame.org/docs/ref/mouse.html) или при помощи эксперимента разберитесь, как же работать с колесиком мыши?

**События по таймеру**

Иногда требуется создавать свои собственные события, которые должны возникать с определенной периодичностью. Например, каждые 10 миллисекунд необходимо проверять значение некоторой переменной, которую могут менять различные обработчики.

Для этого есть следующий механизм:

1. Объявляем свое событие. Это целочисленная константа со значением, которое ранее нигде не использовалось
2. MYEVENTTYPE = 30
3. Вызываем функцию
4. pygame.time.set\_timer(MYEVENTTYPE, 10)
5. Обрабатываем событие в основном цикле игры так же, как и другие стандартные события
6. for event in pygame.event.get():
7. if event.type == MYEVENTTYPE:
8. print("Мое событие сработало")

**Холст (Surface)**

Допустим, мы хотим написать мини-графический редактор.

Ведь каждый программист должен в своей жизни хотя бы раз:

1. Отсортировать массив
2. Написать свой мини-фотошоп
3. Реализовать свой тетрис

Шутка!

Кстати, [Тетрис](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%82%D1%80%D0%B8%D1%81) — вполне хороший итоговый проект по этому модулю.

Кажется, что все совсем просто. Возьмем предыдущий пример, уберем очистку экрана, перенесем строку screen.fill((0, 0, 0)) за цикл — и все! Простейший фотошоп готов!

# очищаем экран один раз в самом начале

screen.fill((0, 0, 0))

while running:

for event in pygame.event.get():

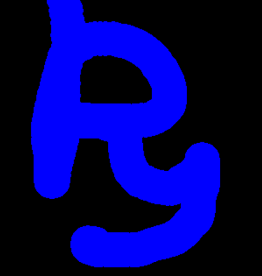
if event.type == pygame.QUIT:

running = False

if event.type == pygame.MOUSEMOTION:

pygame.draw.circle(screen, (0, 0, 255), event.pos, 20)

pygame.display.flip()



Рисовать он может только синим цветом, постоянно и от края экрана, но это легко исправить.

Проблема возникнет в тот момент, когда мы захотим отменить последнее действие или, как в настоящих редакторах, сначала наметить место будущего прямоугольника, а потом уже нарисовать его.

Принципиально есть два решения:

1. Сохранять изображения в виде команд, построив таким образом аналог редакторов векторной графики
2. Рисовать прямоугольник на отдельном холсте и накладывать новый холст на старый. Для этого в классе Surface предусмотрен метод blit(). Два его основных параметра: переменная холста и позиция, куда копировать. Если необходимо, третьим параметром можно указать, какую часть изображения копировать

Реализуем задуманное вторым путем.

Создадим второй холст и будем:

* Копировать второй холст на основной (на экран). Если мы в режиме рисования, то рисовать на экране текущий прямоугольник
* При **нажатии** на кнопку мыши — запоминать начальную вершину и включать режим «рисование»
* При **движении** мыши запоминать ширину и высоту
* При **отпускании** мыши копировать основной холст (экран) на второй холст: фиксировать изменения. И выключать режим «рисование»

screen2 = pygame.Surface(screen.get\_size())

x1, y1, w, h = 0, 0, 0, 0

drawing = False # режим рисования выключен

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

if event.type == pygame.MOUSEBUTTONDOWN:

drawing = True # включаем режим рисования

# запоминаем координаты одного угла

x1, y1 = event.pos

if event.type == pygame.MOUSEBUTTONUP:

# сохраняем нарисованное (на втором холсте)

screen2.blit(screen, (0, 0))

drawing = False

if event.type == pygame.MOUSEMOTION:

# запоминаем текущие размеры

w, h = event.pos[0] - x1, event.pos[1] - y1

# рисуем на экране сохранённое на втором холсте

screen.fill(pygame.Color('black'))

screen.blit(screen2, (0, 0))

if drawing: # и, если надо, текущий прямоугольник

pygame.draw.rect(screen, (0, 0, 255), ((x1, y1), (w, h)), 5)

pygame.display.flip()

На самом деле, мы уже пользовались вторым холстом на первом занятии, когда выводили текст:

font = pygame.font.Font(None, 50)

text = font.render("Hello, Pygame!", 1, (100, 255, 100))

text\_x = width // 2 - text.get\_width() // 2

text\_y = height // 2 - text.get\_height() // 2

text\_w = text.get\_width()

text\_h = text.get\_height()

screen.blit(text, (text\_x, text\_y))

В этом фрагменте переменная text — это тоже холст. Его создает метод render(), а дальше он просто копируется в нужное место методом blit().

**Памятка по решению задач**

При решении задач считайте, что:

* Цвета соответствуют цветам, определенным в Pygame теми же названиями. Например, «желтый» соответствует pygame.Color('yellow')
* Если цвет в условии не указан, считайте цвет фона черным, цвет рисования — белым
* Если толщина линии не указана, считайте, что фигуры должны быть нарисованы закрашенными

[Справка](https://yandex.ru/support/lyceum-students)

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках проекта «Яндекс.Лицей», принадлежат АНО ДПО «ШАД». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «ШАД».

© 2018 – 2020  ООО «[Яндекс](https://yandex.ru/)»

Чаты