Игра 1: “мяч и ворота”

# Установка библиотеки Pygame

## Требования

Для установки pygame вам потребуется Python версии до 3.7 включительно.

Python 2 версии также подойдет.

## Установка python

1. Перейдите по ссылке <https://www.python.org/downloads/release/python-377/>
2. Во вкладке Files выберите нужную вам ссылку
3. Загрузите файл и следуйте инструкциям по установке

## Установка IDE Pycharm

Для комфортной работы с кодом программы рекомендуем установить интегрированную среду разработки (IDE). Такой софт создан для того, чтобы программистам было легче писать код, Pycharm помогает программисту избегать ошибок благодаря удобному графическому интерфейсу и позволяет запускать и отлаживать код прямо внутри среды разработки.

1. Чтобы скачать Pycharm перейдите по ссылке <https://www.jetbrains.com/ru-ru/pycharm/download/>
2. Выберите вашу ОС и загрузите файл.
3. Установите Pycharm, следуя инструкциям по установке.

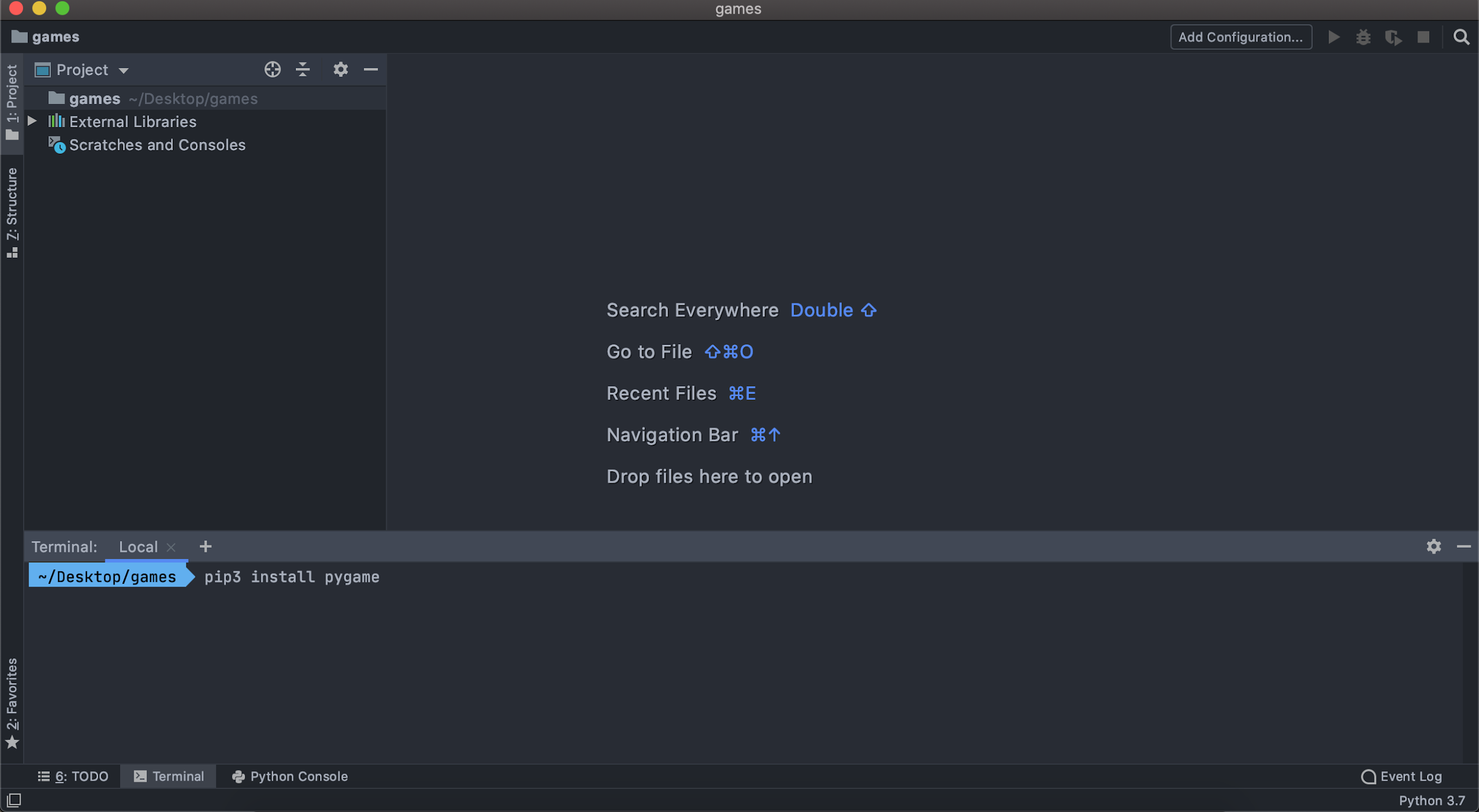
## 

## Настройка Pycharm и установка библиотеки Pygame

Все наши игры будут написаны с помощью библиотеки Pygame, которую следует заранее установить, для этого сначала настроим Pycharm для работы:

1. Создадим папку в удобном нам месте, к примеру, на рабочем столе. Выберем имя, к примеру, “games”.
2. Откроем Pycharm и выберем “Create New Project”.
3. Выберем вашу новую папку “games” и во вкладке “Project Interpreter” убедимся, что выбрана “Existing interpreter”, в котором указана “Interpreter: Python 3.7”.
4. Нажмем “Create”.

Теперь установим библиотеку Pygame:

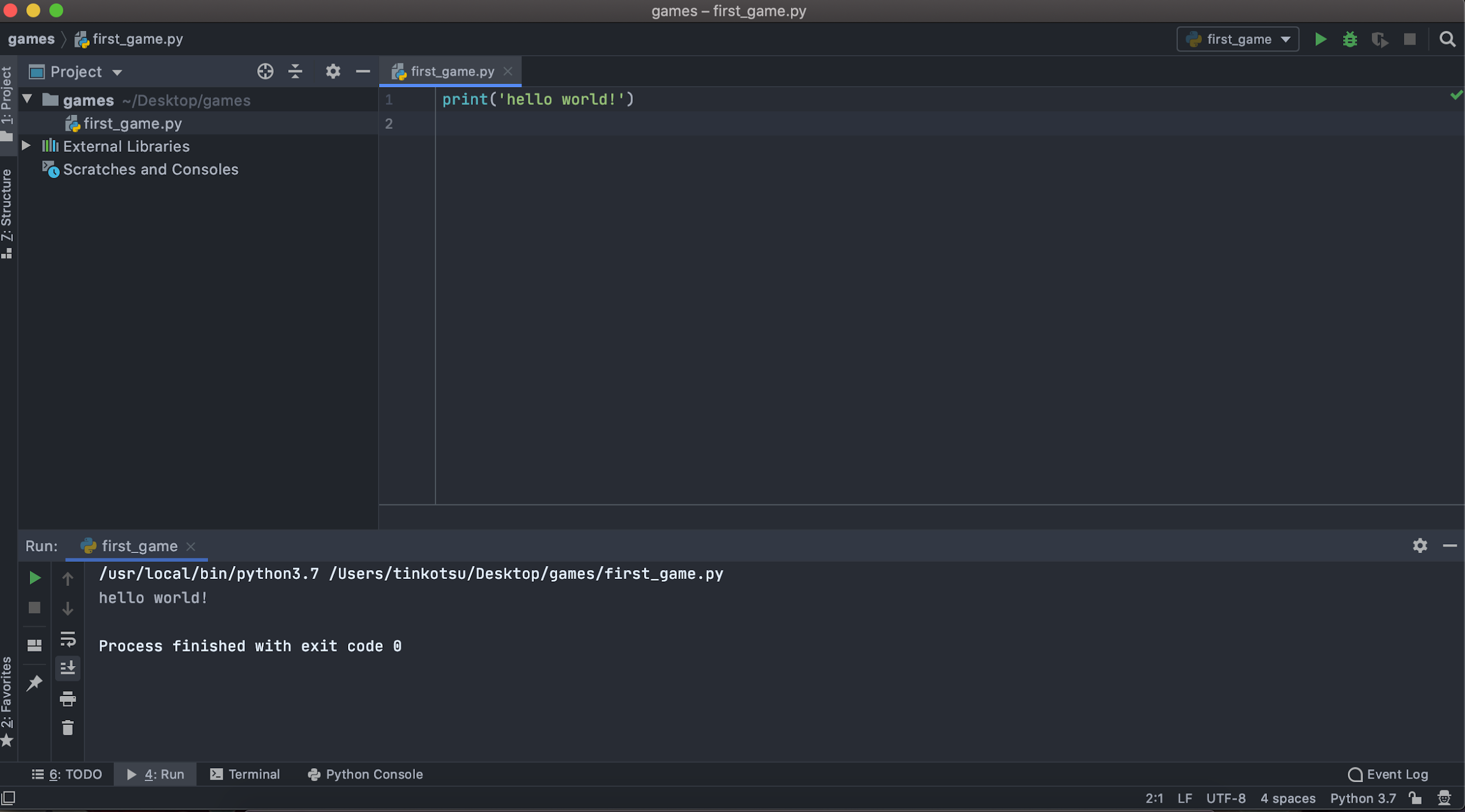
1. Откройте терминал, интегрированный в Pycharm (кнопка расположена слева-внизу интерфейса “Terminal”).
2. Введите команду pip3 install pygame и нажмите enter.
3. Если всё сделали правильно, python должен был установить библиотеку, и теперь мы готовы, чтобы начать разработку игр.

# 

# Приступаем к работе

Для написания игр мы будем использовать среду разработки Pycharm, для начала создадим файл и проверим работоспособность среды:

1. Нажмем правой кнопкой мыши по нашей папке “games”, выберем “New” и в открывшейся вкладке - python file, введем название, к примеру, “first\_game”.
2. Откроем наш файл двойным щелчком мыши по нему и напишем первую строку кода: print(‘Hello, world!’).
3. Нажмем правой кнопкой мыши по свободному месту в редакторе и выберем “Run ‘first\_game’”, так мы будем каждый раз запускать нашу программу.
4. Все должно было получиться и в консоли вывода отобразилась наше приветствие:



# 

# 

# 

# 

# 

# 

# 

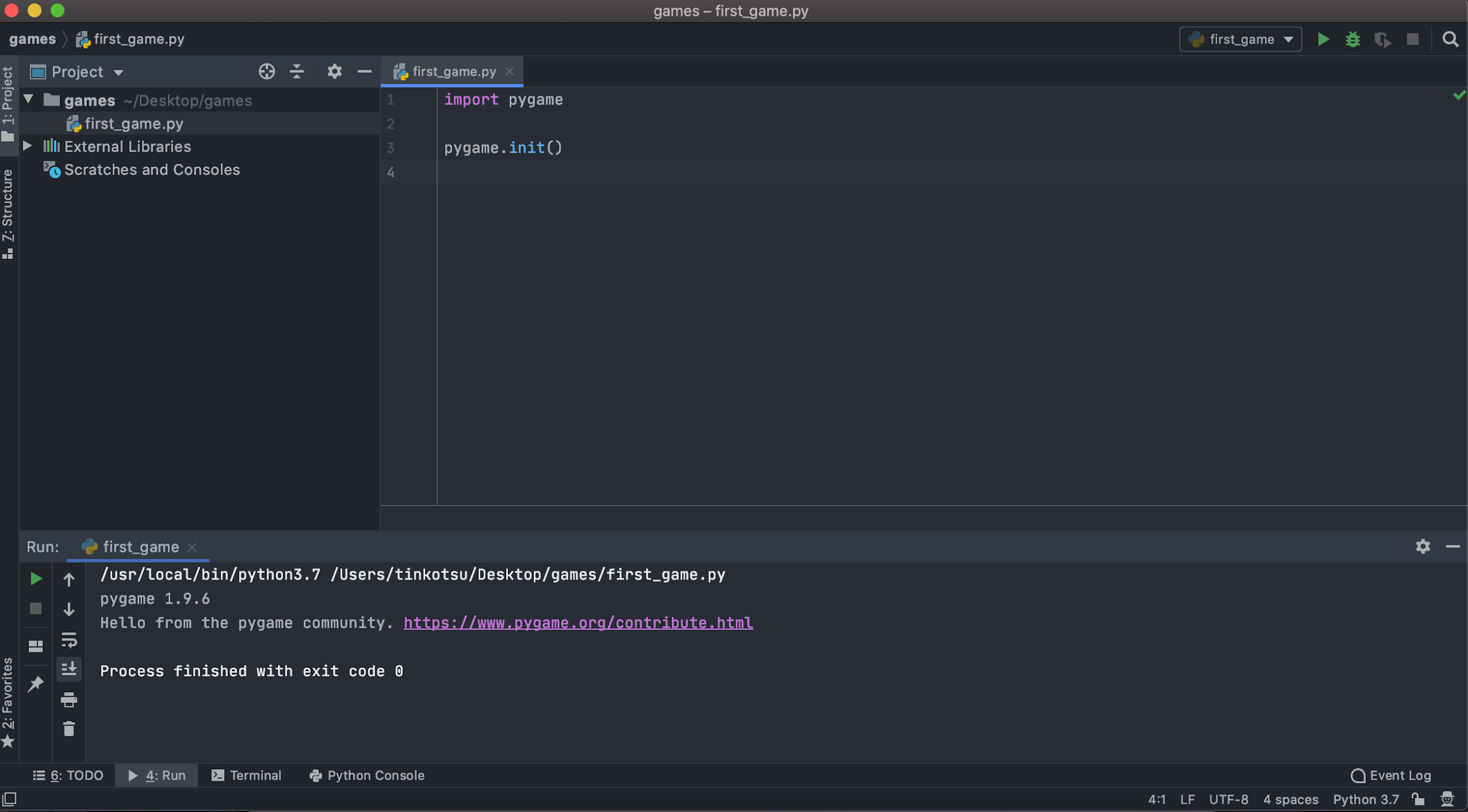
# Начало программы

## Документация Pygame

Мы уже знакомы с таким понятием как “документация”. И у Pygame есть своя документация с актуальным списком всех классов, модулей и методов. Найти её можно по ссылке <https://www.pygame.org/docs/>

## Импорт и инициализация

Каждый раз начиная писать игру на библиотеке Pygame, мы будем импортировать эту библиотеку в начале нашей программы, для этого:

1. Очистим наш редактор от лишних строк кода.
2. Напишем строку import pygame в первой строчке программы, после чего отступим одну пустую строку.
3. В третей строке напишем pygame.init()- эта строка инициализирует все модули внутри библиотеки и также необходима в начале программы.
4. Теперь мы импортировали все нужные модули из библиотеки pygame и инициализировали параметры для работы с ними.
5. Давайте запустим программу и проверим, что все работает, окно Pygame должно появиться и сразу закрыться, а в консоли вывода мы увидим приветствие от команды Pygame.
6. Отлично! Теперь начнем дополнять наш код всем необходимым, чтобы получилась первая игра.

# Окно игры

Как и в любом другом скрипте, написанном на python, при создании игры мы будем работать с объектами. Первый и один из самых важных объектов - это объект нашего рабочего окна, экрана, на котором будут отображаться графические элементы - важнейшая составляющая любых игр.

Пора создать окно игры:

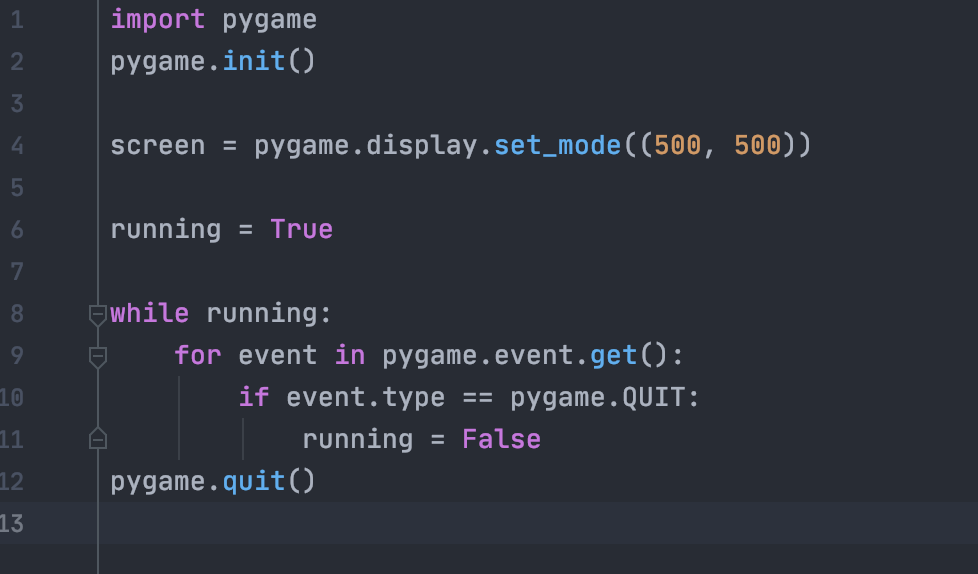
1. Создадим объект и присвоим его переменной screen, для этого напишем строчку screen = pygame.display.set\_mode([500, 500])
2. На первый взгляд может показаться, что строчка получилась очень сложной, но на самом деле всё просто:
   1. Мы обращаемся к классу display внутри класс pygame и вызываем у display модуль set\_mode, который устанавливает нужный нам размер окна и возвращает объект для работы с ним. Поэтому мы присвоили это выражение переменной screen.
   2. В метод set\_mode мы подали список из двух параметров: [500, 500], где первое число - это ширина, а второе - высота окна. Попробуй изменить значения на другие, например, напишем стандартное разрешение экрана [800, 600]. Вместо списка можно было использовать кортеж.
3. Давайте попробуем запустить программу и увидим, что снова окно появилось и сразу же закрылось, все правильно - ведь мы не указали программе работать какое-то время, создался объект окна после чего программа закончила свое выполнение. Для решения этой небольшой проблемы нам нужно использовать бесконечный цикл.

# Бесконечный цикл

Для того, чтобы наша программа работала до того момента, пока пользователь или игрок сам не решит выйти, нам нужно использовать бесконечный цикл, давайте разберемся как это сделать:

1. Создадим переменную для выхода из цикла и присвоим ей значение True. Пока эта переменная имеет значение True, наша окно будет активно, как только переменной присваивается значение False, окно игры должно немедленно закрыться.
2. К примеру, назовем нашу переменную running, тогда напишем строчку

running = True. Пока running - истина (True) цикл будет выполняться, а когда пользователь захочет выйти (нажмёт на крестик для закрытия игры), переменной присвоим значение False. Давайте напишем это и разберемся с кодом программы.



В 6 строчке программы мы создали переменную, а в 8 строчке запустили цикл, о котором написано выше. В строке 11 мы присваиваем значение переменной running = False. После чего произойдет выход из нашего цикла, и программа перейдет к 12 строчке, где мы вызвали метод quit() у класса pygame, который закрывает все процессы в классе pygame.

Осталось разобраться что же происходит в строках 9 и 10:

1. Как мы и хотели, программа теперь научилась реагировать на нажатие крестика для закрытия, для этого мы используем метод get() у модуля event. get() возвращает нам список всех текущих ивентов, которые создаются в нашей программе, о них мы поговорим позднее, но сейчас нужно знать, что нажатие на крестик - это тоже ивент (событие), которое совершает игрок.
2. Мы проходимся по всему списку ивентов с помощью цикла for и проверяем, является ли какой-либо активный ивент нажатием на крестик, для этого мы используем условие в 10 строке. Атрибут type у event определят тип ивента, его мы сравним c pygame.QUIT - это тип ивента, который создается, если пользователь нажал на крестик.
3. Все просто! Если условие выполнилось, мы присваиваем новое значение переменной, от которой зависит наш цикл, и выходим из программы.

Давайте попробуем запустить нашу программу.  
Все получилось и теперь программа не закрывается через секунду, а ждет, пока мы не нажмем на крестик. Отлично, мы готовы продолжать!

# Геометрические фигуры

Все игры начинаются с чего-то простого: гоночную машину в начале разработки легче изобразить как движущийся прямоугольник и т.д. Так и наша первая игра будет содержать в себе простейшие графические элементы, давайте разберемся, как их использовать:

1. Для отрисовки фигур мы будем использовать модуль pygame.draw. Начнем с функции rect(). Давайте напишем эти три строчки перед нашим циклом:

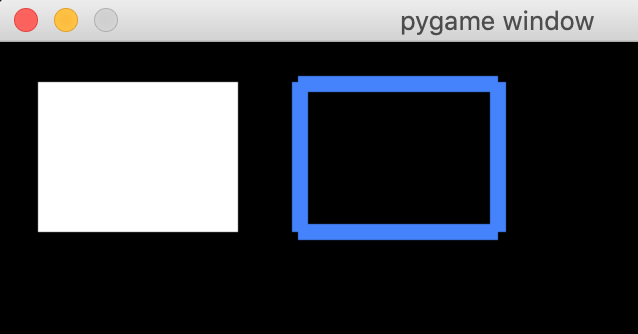
|  |
| --- |
| pygame.draw.rect(screen, (255, 255, 255), (20, 20, 100, 75))  pygame.draw.rect(screen, (64, 128, 255), (150, 20, 100, 75), 8)  pygame.display.flip() |

После того, как мы передали какую-либо графику на нашу поверхность (screen), мы будем использовать метод flip() (или метод update()) в модуле display. Этот метод необходим для того, чтобы увидеть изменения на экране. Его мы вызвали в третьей строке примера выше.

Первый аргумент в функциях рисования всегда поверхность, на которой размещается фигура - наш screen, который мы определили выше.

Вторым аргументом всегда передается цвет в формате RGB - целочисленный кортеж из трех элементов. Например, (255, 0, 0) определяет красный цвет.

Далее идут специфичные для каждой фигуры параметры, последним аргументом скорее всего является толщина контура. Так во второй строчке мы определили толщину контура 8, прямоугольник отобразится незаполненным, а цветом будут закрашены его стенки.

Последний аргумент в первой строчке и предпоследний во второй - это прямоугольные области, они определяют экземпляр класса Rect. Прямоугольная область имеет 4 параметра: координату x и координату y верхнего левого угла, ширина и высота прямоугольника.

Для повышения читаемости кода можно:

1. Вынести прямоугольную область (экземпляр класса Rect) в отдельную переменную и после этого передавать ее в модуль для отрисовки.
2. Обозначить цвета переменным в начале программы для того, чтобы эффективно их использовать в программе.

|  |
| --- |
| 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(screen, WHITE, (20, 20, 100, 75))  pygame.draw.rect(screen, LIGHT\_BLUE, r1, 8)  pygame.display.flip() |

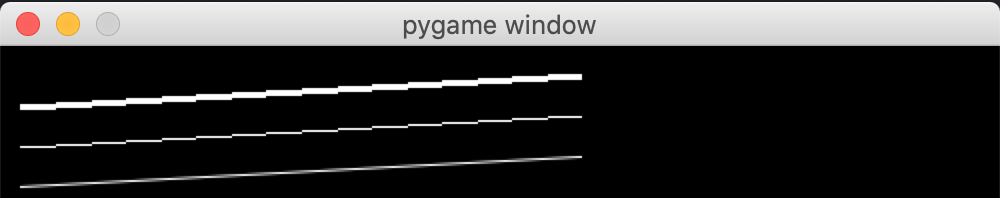
Попробуй нарисовать незаполненные и сплошные прямоугольники разного цвета в своем окне.

Давайте разберемся с остальными фигурами:

## Линия

А точнее отрезок. Чтобы его нарисовать, нужно указать координаты его концов. line() рисует обычную линию; aaline() - сглаженную, толщину для нее указать нельзя.

|  |
| --- |
| pygame.draw.line(screen, WHITE, [10, 30], [290, 15], 3)  pygame.draw.line(screen, WHITE, [10, 50], [290, 35])  pygame.draw.aaline(screen, WHITE, [10, 70], [290, 55]) |

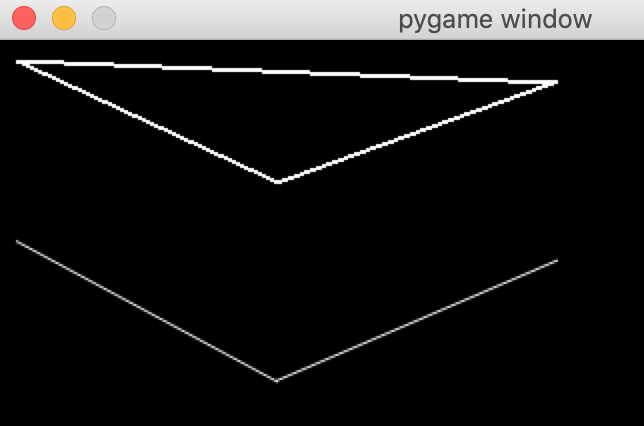


## Ломаные линии

Функции lines() и aalines() рисуют ломаные линии

|  |
| --- |
| pygame.draw.lines(screen, WHITE, True, [[10, 10], [140, 70], [280, 20]], 2)  pygame.draw.aalines(screen, 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.

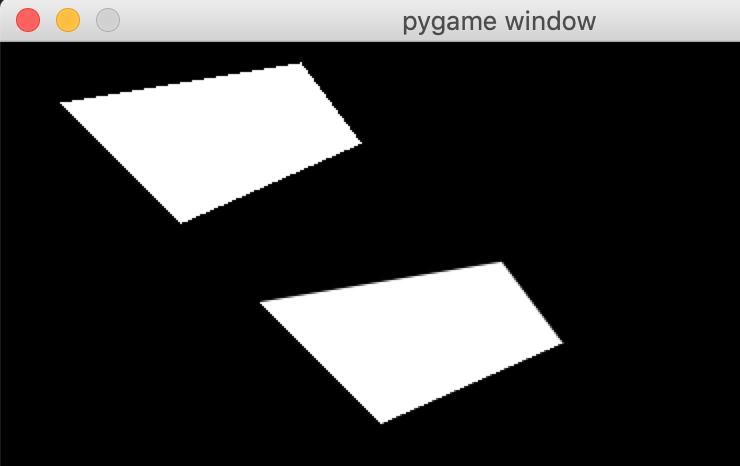


## Многоугольник

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

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

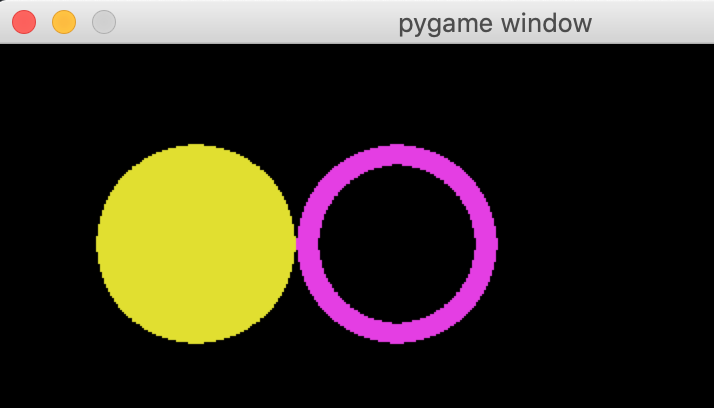
Сглаженная ломаная здесь повторяет контур второго многоугольника, чем сглаживает его ребра.

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

## Круг

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

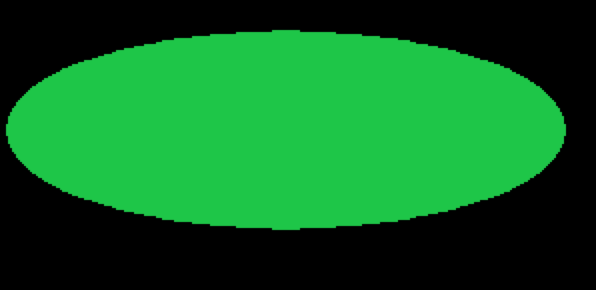
|  |
| --- |
| pygame.draw.circle(screen, YELLOW, (100, 100), 50)  pygame.draw.circle(screen, PINK, (200, 100), 50, 10) |



## Эллипс

Эллипс можно нарисовать с помощью функции ellipse(). Аргументами передается прямоугольная область, которая описывает его:

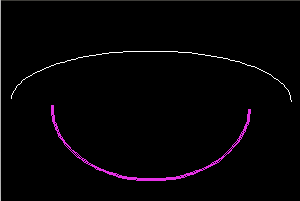
|  |
| --- |
| pygame.draw.ellipse(screen, GREEN, (10, 50, 280, 100)) |



## Дуга

Нарисовать можно с помощью функции arc().

|  |
| --- |
| 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.

# 

# 

# 

# Анимация

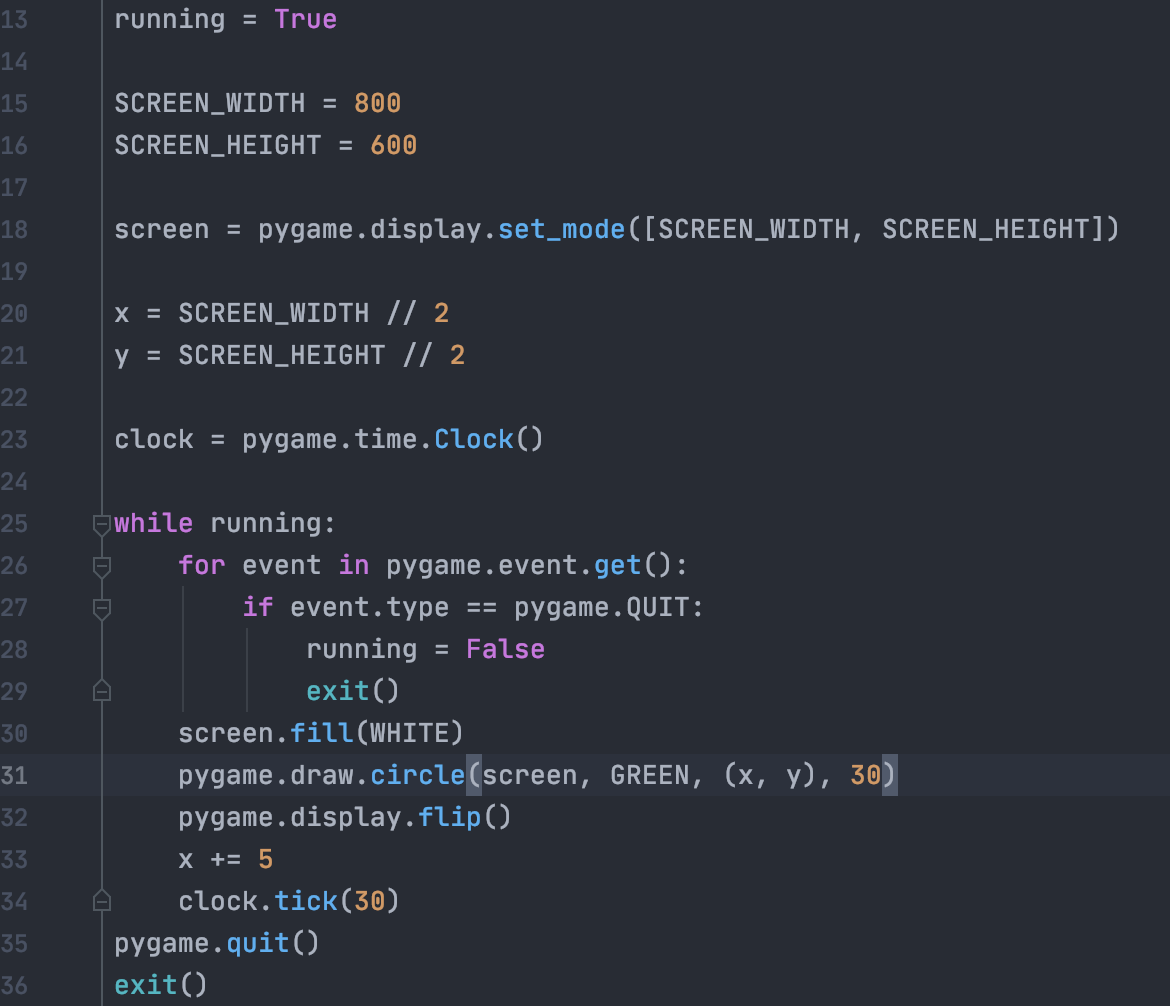
На самом деле объекты не передвигаются на экране. Чтобы понять, как двигается предмет, давайте представим движение одного пикселя на экране:

1. Нарисуем пиксель на координатах (10, 10).
2. Переместим его выше на координату (10, 9), чтобы это сделать нам необходимо закрасить пиксель на координатах (10, 10) цветом фона и нарисовать новый на координатах (10, 9).
3. Пользователю показалось, что пиксель переместился, но на практике мы стерли старый и отрисовали новый.
4. На этом принципе построена анимация в играх, и подобный алгоритм мы будем использовать в дальнейшем.

Давайте попробуем нарисовать круг и заставить его двигаться на экране, для этого:

1. Для удобства использования ширины и высоты экрана, запишем эти значения в соответствующие переменные SCREEN\_WIDTH и SCREEN\_HEIGHT. Присвоим им необходимые значения и в методе set\_mode() используем кортеж, состоящий из этих двух переменных.
2. Создадим переменные x и y, которые будут отвечать за положение нашего круга на экране. Присвоим им стартовое значение - середина экрана, для этого возьмем половину от ширины экрана и поместим это значение в x, а половину высоты - в y (нам необходимо использовать целочисленное деление “//”, потому что координаты принимаются только в целом виде, а деление на два не всегда осуществляется без остатка). Мы используем эти значения для того, чтобы наш кружок начал свое движение с центра экрана.
3. В нашем бесконечном цикле обозначим по порядку:
   1. Заполнение всего экрана белым цветом, для этого используем метод fill у нашего объекта screen.
   2. Нарисуем круг по текущим координатам x и y.
   3. Отобразим все объекты на экране с помощью метода flip().
   4. Изменим координату x нашего круга, чтобы при следующем проходе, круг отобразился на новом месте. Величина, на которое меняем x и есть скорость движения круга.
4. Еще один момент про скорость движения: мы задали то, на сколько пикселей будет сдвигаться объект при каждой итерации, но мы не указали скорость смены этих итераций. Изначально этот параметр зависит от частоты процессора и на разных компьютерах круг может двигаться с разной скоростью, мы бы этого не хотели. Для этого мы установим ограничение по кадрам в секунду (FPS), многие, наверное, слышали этот термин. Как это сделать:
   1. Перед циклом добавим строчку clock = pygame.time.Clock(), это нам позволит создать объект clock, с помощью которого мы будем регулировать ограничение кадров в секунду.
   2. В конец нашего цикла добавим строчку clock.tick(30) - метод tick() устанавливает ограничение в конце каждой итерации на частоту смены кадров.

Если все сделали правильно, то у вас должно было получиться что-то похожее на мой вариант (выше 13 строки расположен импорт, инициализация и обозначение цветов):



Давайте запустим и посмотрим на результат.

Здорово! Наш круг начал двигаться, но он очень быстро достиг края экрана и скрылся за ним. Мы бы хотели, чтобы наш круг не улетал за границы, а отскакивал от них, давайте разберемся с тем, как это сделать:

1. Решим эту задачку с помощью логики и представления о движении круга. Если мы хотим, чтобы кружок отталкивался от края окна, то нам нужно менять направление движения каждый раз, когда наш круг касается края.
2. Создадим отдельную переменную для нашей скорости. К примеру, v. Перед циклом присвоим ей значение 5. Теперь в цикле мы будем прибавлять не 5, а нашу переменную v.
3. Создадим отдельную переменную для радиуса нашего круга. К примеру, r. Перед циклом присвоим ей значение 30. Теперь при отрисовке круга будем использовать эту переменную, она нам понадобится для более точной настройки отскока.
4. Добавим одно единственное условие в наш цикл. Давайте подумаем о том, какое это должно быть условие. Если центр нашего круга (наша переменная x) сместился правее края экрана, то круг уже наполовину зашел за край. Мы хотим, чтобы как только одна из крайних точек круга касалась края экрана, он тут же менял свое направление. Это не сложно реализовать, ведь мы знаем, что все крайние точки круга расположены от центра на расстоянии радиуса (наша переменная r). Тогда если сумма координаты центра круга (x) + значение радиуса ( r ) стало больше ширины экрана, значит нашему кругу пора ехать в другое направление (скорость становится не положительной, а отрицательной: v = -v). Также если координата центра круга расположена на расстоянии r от левой границы экрана (0), нам снова нужно поменять направление движения (v = -v). Тогда наше условие будет выглядеть так:

|  |
| --- |
| if x + r > SCREEN\_WIDTH or x - r < 0:  v = -v |

Добавим условие в конец цикла перед строк, ограничивающей частоту кадров.

Давайте еще раз разберемся с этим выражением:

Если круг оказался правее правой границы экрана или левее левой границы, мы меняем вектор движения на противоположное значение (Если круг двигался вправо и достиг края экрана, его координата увеличивалась и скорость была положительной, теперь нам нужно уменьшать координату и двигать круг влево. Если круг двигался влево и достиг края экрана, его координата уменьшалась и скорость была отрицательной, теперь нам нужно увеличивать координату и двигать круг вправо. Это все делается умножением скорости на -1 или же обычным присвоением v = -v).

Запустите программу и убедитесь, что все работает.

## Задание

Попробуй изменить поведение движения круга таким образом, чтобы он передвигался не вправо-влево, а вверх-вниз и также сделай отскоки от границ окна. Попробуй поменять скорость движения круга. Подумай, как бы мы могли сделать так, чтобы круг двигался не по одной прямой, а, например, по диагонали?

# 

# Отображение текста

Для работы со шрифтами и текстом мы будем использовать метод pygame.font. 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('/usr/share/fonts/truetype/msttcorefonts/Arial.ttf', 36) |

Второй аргумент - это размер шрифта в пикселях.

Узнать, какие шрифты есть в системе, можно с помощью функции get\_fonts().

Вы можете скопировать шрифт в каталог программы и обращаться к нему без адреса, только по названию файла:

|  |
| --- |
| 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, 36) |

Этот вариант подойдет для нашей игры.

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 (0 – нет, 1 – есть), третьим – цвет текста. При необходимости четвертым аргументом можно указать цвет фона.

|  |
| --- |
| f1 = pygame.font.Font(None, 36)  text1 = f1.render('Hello world', 1, GREEN)  screen.blit(text1, (10, 50)) |

Метод blit() позволяет разместить переданную поверхность (в данном случае text1) на указанных координатах.

Добавьте эти 3 строчки внутрь нашего цикла после заполнения экрана белым цветом, теперь в левом верхнем углу будет надпись ‘Hello world’.

## Выравнивание текста

Для нашей будущей игры будет очень кстати, чтобы текст не висел в левом верхнем углу, а всегда размещался по центру, независимо от сообщения.

1. Как мы уже поняли, метод render() у объекта шрифта создает поверхность с сообщением, которое мы накладываем поверх нашего экрана с помощью метода blit() в заданных координатах. Важно понимать, что метод blit() накладывает поверхность с сообщениям на наш экран таким образом, что переданные координаты - это координаты левого верхнего угла сообщения. Как же нам тогда размещать сообщение по центру?
2. Для этого мы используем метод get\_rect(), который создает объект типа Rect, размеры которого соответствуют размерам нашего объекта (text), а сам новый объект является прямоугольной областью. У нового объекта есть свойства-координаты в отличие от старого, это нам и нужно. Модифицируем наши последние изменения таким образом:

|  |
| --- |
| f1 = pygame.font.Font(None, 36)  text1 = f1.render('Hello world', 1, GREEN)  place = text.get\_rect(center=(SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2))  screen.blit(text, place) |

Мы создали переменную place, которой присвоили результат работы метода get\_rect() - прямоугольную область. В метод мы передали значения флага center - кортеж из координат: (середина ширины экрана, середина высоты экрана). Теперь place обзавелась такими координатами, что её центр - это центр нашего окна. При передаче объекта place в метод blit() вторым аргументом, оттуда автоматически выбираются координаты левого верхнего угла прямоугольной поверхности. Координаты, в свою очередь, автоматически подставились при создании place, когда мы указали, что центр этого объекта должен быть по координатам, которые находятся в центре экрана.

1. Почему бы нам не создать отдельную функцию для отображения текста, в которую мы будем принимать сообщение (message) и цвет, которым необходимо отобразить это сообщение (m\_color):

|  |
| --- |
| font = pygame.font.Font(None, 36)  def put\_text(message, m\_color):  text = font.render(message, True, m\_color)  place = text.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT // 2))  screen.blit(text, place)  pygame.display.flip() |

Теперь каждый раз при вызове этой функции, мы будем отображать переданное сообщение message цветом m\_color в центре экрана.

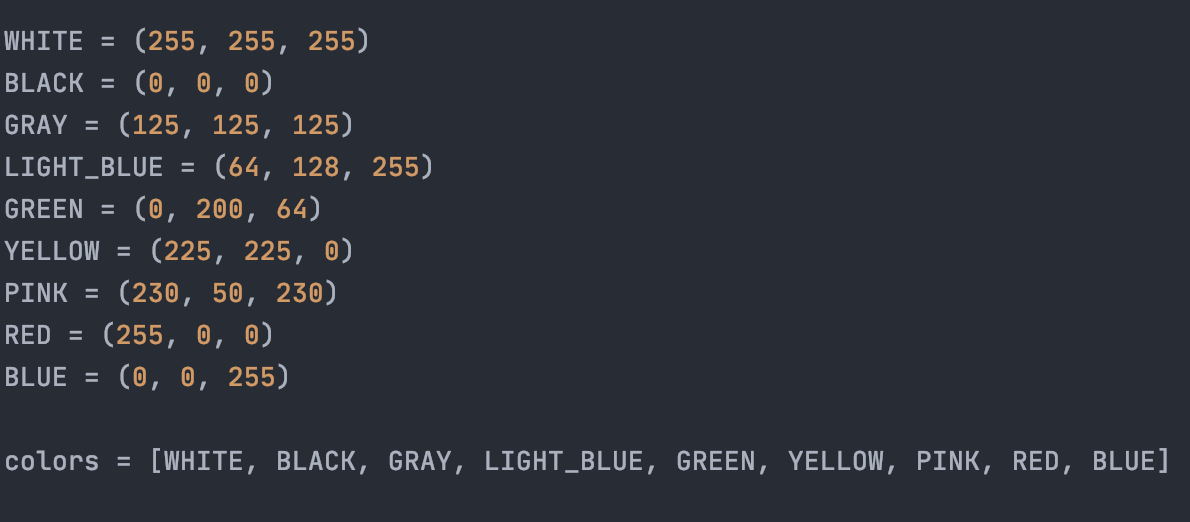
# Нажатие клавиш

Для нашей игры потребуется еще один момент: программа должна понимать, когда игрок нажал на клавишу. Для этого нам познакомиться с событиями клавиатуры. Подробнее о них мы поговорим при создании игр посложнее, но сейчас нам нужно знать то, что:

1. События клавиатуры создаются при каком-либо действии игрока. Они помещаются в тот же список, по которому мы проходим циклом for, когда смотрим, не нажал ли пользователь на крестик.
2. В данном случае типом события (event.type) будет событие pygame.KEYDOWN, который отвечает за нажатие какой-либо клавиши.
3. У события event также есть атрибут ‘key’, который отвечает за то, какую именно клавишу нажал пользователь.

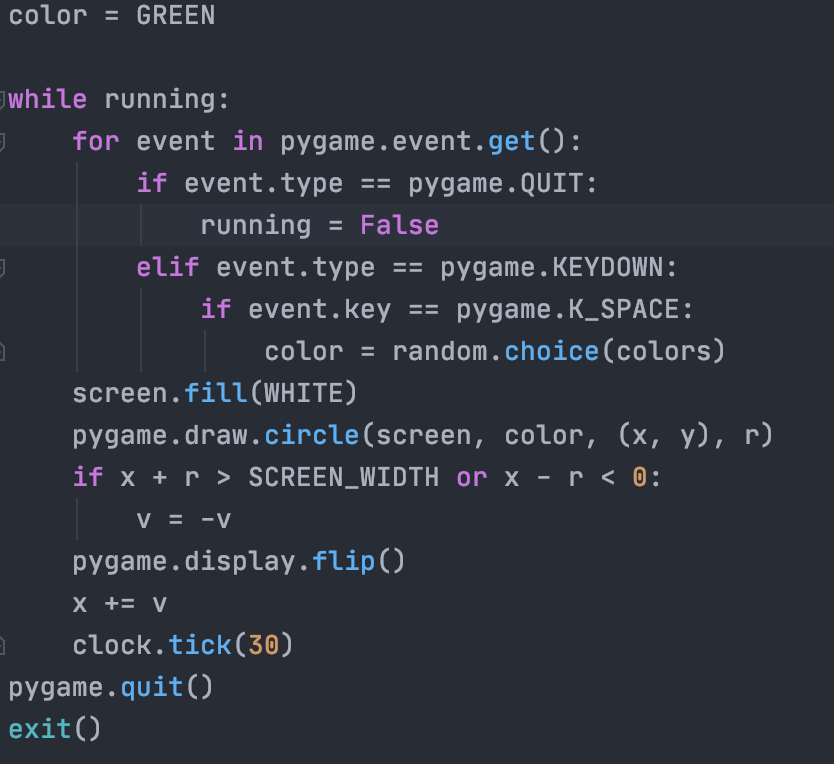
Давайте посмотрим, как это можно применить на практике:

1. Создадим список цветов из уже созданных наименований:



1. Импортируем модуль random в самом начале программы.
2. Создадим переменную color, и присвоим ей значение GREEN до цикла. Это будет начальный цвет нашего круга, а внутри метода draw.circle() поменяем значение цвета на нашу переменную color.
3. Добавим условие, что при нажатии на пробел, color будет меняться на рандомный из нашего списка colors.

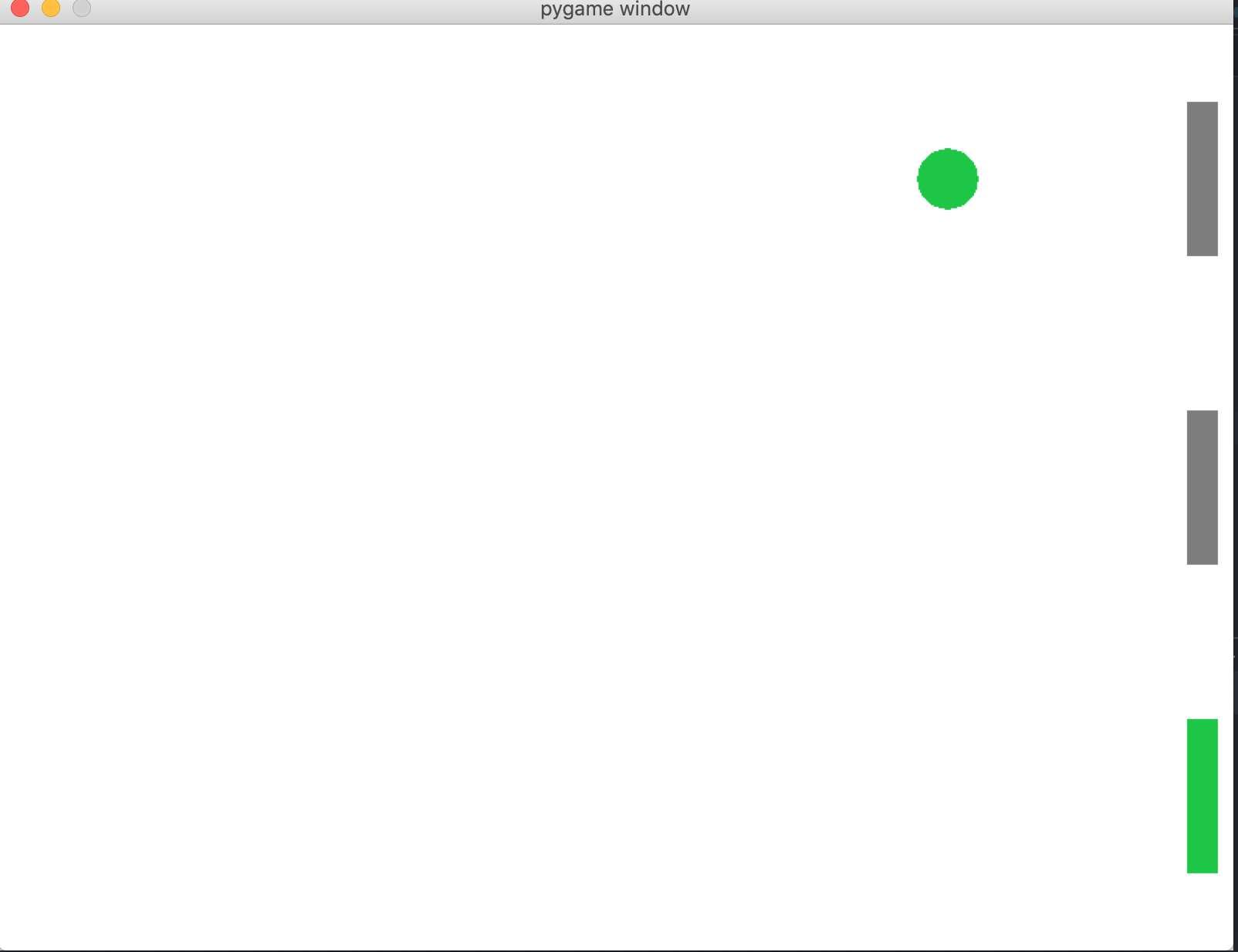
Теперь наш основной цикл выглядит таким образом:



Попробуй нажать на пробел   
и посмотри, как изменится цвет круга. Здорово, не правда ли?

# Цель игры

Игра, которую мы будем создавать сегодня достаточно простая, но чтобы ее написать, нам понадобятся все элементы, которые мы изучили выше. Выглядеть она будет примерно так:



На экране мы видим кружок и трое “ворот”.

Цель игры - активировать одни из ворот нажатием клавиши с 1 по 3. Необходимо успеть это сделать до того, как кружок долетит до одних из ворот.

Обозначим ограничение в 10 уровней, и каждый новый уровень скорость перемещения круга будет увеличиваться. Ближе к последнему уровню будет уже не так просто успеть активировать какие-либо ворота до того, как кружок достигнет их.

Игра на скорость реакции и внимательность.

На скриншоте показан пример, когда игрок нажал цифру 3 и активировал нижние ворота. Сейчас, если он не успеет нажать цифру 1, он проиграет, ведь кружок залетит в неактивированные ворота.

# 

# Реализация

## План

Уточним последовательность нашей игры:

1. Кружок появляется рандомно напротив одних из трех ворот.
2. Пока кружок летит до ворот у игрока есть возможность успеть активировать нужные ворота. Оставляем возможность поменять выбор, если игрок ошибся. Не будем добавлять хардкора в самую первую игру.
3. Спрашиваем у игрока, хочет ли он продолжить, и если да, то начинаем новый уровень, увеличиваем скорость кружка и переходим к шагу 1.
4. Если пройден десятый уровень, то выводим сообщение о победе.
5. Если кружок влетел в неактивированные ворота, то выводим сообщение о проигрыше.

Вот и все! Мы готовы приступать к написанию кода нашей игры.

## Пишем код

### Переменные

Мы уже имеем некоторые переменные в нашем коде:

* running отвечает за прерывание цикла и выхода из игры, изначально  
  running = True
* x и y отвечают за координаты круга на нашем экране, изначально x = 0
* v - скорость передвижения круга, изначально v = 5
* color - цвет круга, изначально color = GREEN
* добавим переменную lvl, которая будет отвечать за номер текущего уровня, изначально lvl = 1
* добавим переменную won, которая будет отвечать за то, успел ли игрок выбрать правильные ворота до конца уровня, изначально won = 0
* добавим переменную gate, которая будет отвечать за то, какие ворота необходимо активировать - раскрасить зеленым цветом при нажатии соответствующей кнопки, изначально gate = -1, все ворота будут не активированы и раскрашены серым цветом.

Вы могли заметить, что мы не указали стартовое значение переменной y. Сделал я это нарочно, нам понадобится немного рандома, чтобы определить её значение. Как мы уже определились до этого, каждый уровень кружок должен появляться на одной из трех позиций, эта позиция будет выбрана случайна:

1. Создадим список heights, в который положим значения для стартовых высот кружка. Первой позицией будет точка сверху с отступом 100 пикселей от верхнего края, второй позицией будет середина высоты экрана (в этом случае кружок будет лететь по центру), третьей позицией будет точка снизу с отступом 100 пикселей от нижнего края.

|  |
| --- |
| heights = [100, SCREEN\_HEIGHT // 2, SCREEN\_HEIGHT - 100] |

1. Присвоим переменной y перед нашим циклом рандомное значение из этого списка:

|  |
| --- |
| y = random.choice(heights) |

1. Каждый новый уровень мы будем повторять рандомный выбор высоты из списка heights.

### Нарисуем фигуры

При изменении положения нашего кружка, мы должны заполнять весь экран белым цветом и потом заново отрисовывать все фигуры, поэтому логично будет сделать отдельную функцию для отрисовки всех фигур draw(), которую мы будем вызывать в нашем цикле перед тем, как поменять координату x нашего кружка.

|  |
| --- |
| def draw():  screen.fill(WHITE)  pygame.draw.rect(screen, GRAY, (SCREEN\_WIDTH - 20, heights[0] - 50, 20, 100))  pygame.draw.rect(screen, GRAY, (SCREEN\_WIDTH - 20, heights[1] - 50, 20, 100))  pygame.draw.rect(screen, GRAY, (SCREEN\_WIDTH - 20, heights[2] - 50, 20, 100))  pygame.draw.circle(screen, color, (x, y), 20)  pygame.display.flip() |

Давайте разберемся с каждой строкой в функции:

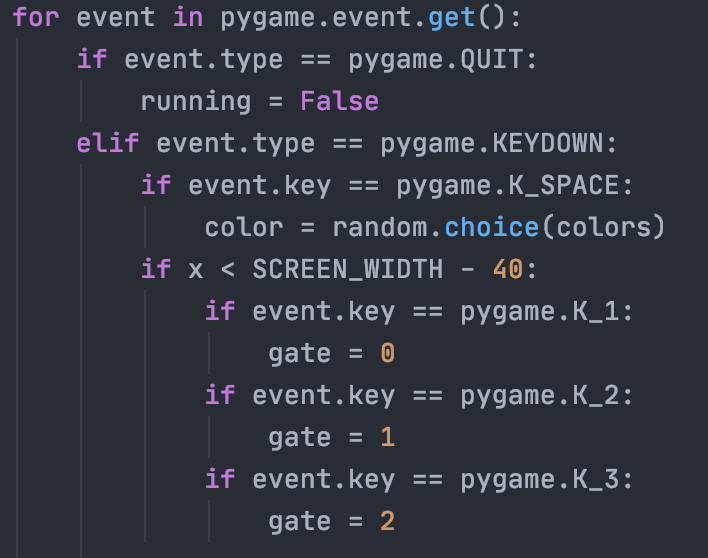
1. Заполним все наше окно белым цветом с помощью метода fill().
2. Нарисуем трое врат серым цветом. Мы помним, что метод rect() принимает координаты левого верхнего угла прямоугольника, поэтому при ширине 20 пикселей, отступим от правого края на значение ширины. Также мы хотим, чтобы кружок летел ровно по центру высоты каждых врат, поэтому при высоте 100 пикселей, мы отступим от каждой высоты (heights[0] = 100, heights[1] = SCREEN\_HEIGHT // 2, heights[2] = SCREEN\_HEIGHT - 100) 50 пикселей, что будет также половина высоты нашего прямоугольника.
3. Нарисуем наш круг выбранным цветом (мы все еще можем менять цвет на пробел) в заданных координатах, радиусом 20.
4. Вызовем метод flip(), чтобы отобразить все изменения на экране.

### Активация ворот

Мы решили, что ворота будут активироваться по нажатию клавиш 1, 2 и 3. Нам необходимо сделать так, чтобы:

1. Активация врат была возможна пока кружок не долетел до начала врат.
2. При активации одних врат, другие должны снова закрашиваться серым, активными могут быть только одни ворота.

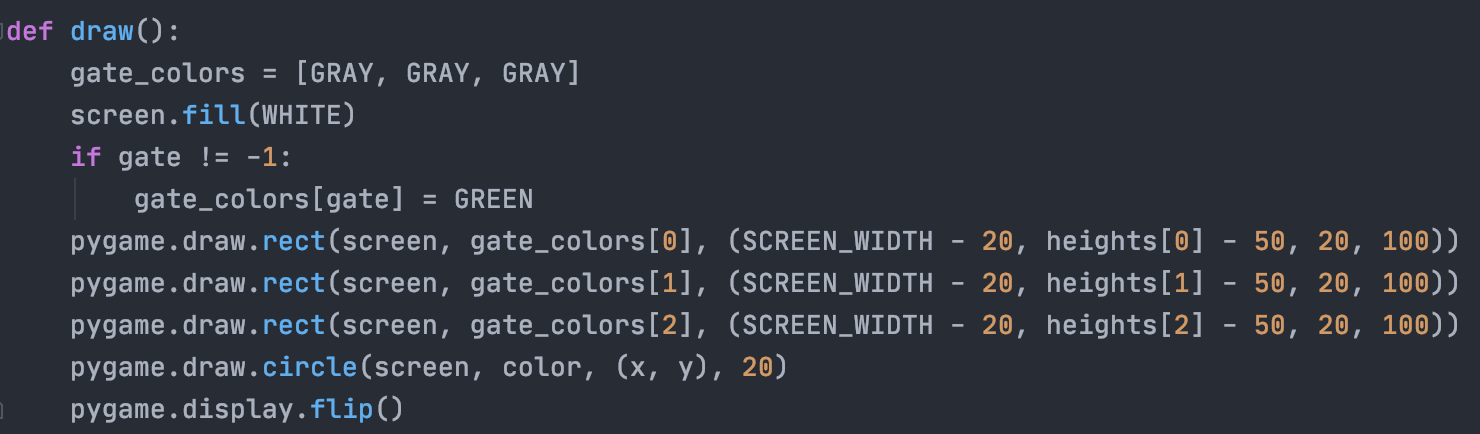
Задачу №1 можно решить, добавив одно условие в место, где мы будем захватывать нажатие клавиш. Теперь оно будет выглядеть так:



Мы добавили условие, что если координата круга x меньше, чем ширина экрана - 40 пикселей, то нажатия клавиш меняют переменную gate, которая отвечает за то, какие ворота необходимо раскрасить зеленым цветом. Такое значение выбрано по причине того, что радиус круга = 20 и ширина врат = 20, значит, что круг коснется врат, когда x будет на расстоянии 40 пикселей от правого края экрана. В этот момент игрок уже не сможет менять свой выбор.

Теперь давайте решим задачу №2:

В переменной gate мы храним порядковый номер врат, начиная с нуля. Это сделано не просто так. Вернемся к нашей функции draw(), как мы помним, она вызывается на каждой итерации нашего главного цикла. Немного модифицируем её таким образом:



Теперь при каждом вызове функции, мы будем локально создавать список gate\_colors, который изначально заполнен тремя элементами, отвечающими за серый цвет. Изначально наша переменная gate = -1, это значит, что все врата должны быть нарисованы серым цветом, поэтому мы добавили условие, что если только мы поменяли нашу переменную gate на какое-либо число (смотрим в решение задачи №1 чуть выше), то этот элемент списка цветов станет зеленого цвета - активируется. После чего мы отрисуем трое врат, а вместо серого цвета обратимся к элементу списка цветов: для первых врат - к gate\_colors[0], для вторых - к gate\_colors[1], для третьих - к gate\_colors[2]. Таким образом, если мы активируем сначала верхние врата, они поменяют цвет на зеленый, потом если мы активируем средние врата, верхние закрасятся серым, а средние станут зеленого цвета. Как мы и хотели.

### Проверка победы

Когда наш кружок коснулся каких-либо ворот, нам нужно узнать, правильные ли ворота были активированы. Для этого в главном цикле создадим условие, что если кружок коснулся врат, то:

1. Остановим движение кружка. Так будет нагляднее для игрока.
2. Сравним высоту кружка и высоту активированных врат (если какие-либо врата успели активировать).
3. Выведем сообщение о проигрыше или о том, что уровень пройден, и предложим перейти к следующему.

Выглядеть это будет таким образом:

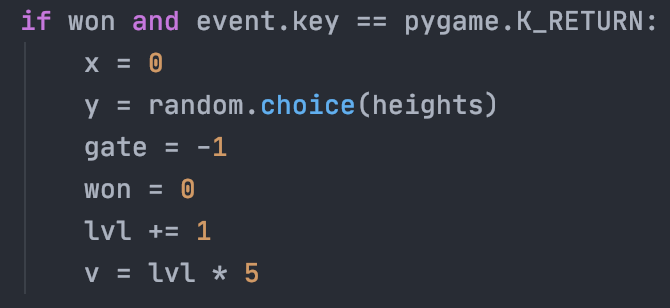


Давайте разбираться:

1. Если кружок коснулся, значит его координата больше или равна отступу врат + радиусу круга (20 + 20).
2. Остановим движение круга, скорость v = 0.
3. Проверим, если игрок не успел активировать врата, то gate == -1 или heights[gate] не равно y. Чтобы понять это неравенство нужно вспомнить, что y каждый уровень принимает рандомное значение из списка heights, а gate может принимать значение 0, 1 или 2, тогда heights[gate] при gate = 0 будет равно heights[0] - а эта позиция для кружка, который летит сверху. Если эти два значения не равны, тогда игрок активировал неверные врата.
4. Создадим сообщение msg и выберем соответствующий цвет msg\_color.
5. Если игрок победил, сохраним это в переменную won.
6. Выведем сообщение на экран с помощью нашей функции put\_text().

### Победа в уровне

Если уровень пройден, мы предлагаем игроку перейти на следующий, когда он будет готов и нажмет Enter. Нам останется схватить нажатие клавиши и обновить все переменные, которые участвуют в нашей игре, сделать это мы должны в нашем цикле for, который проходит по списку всех активных событий:



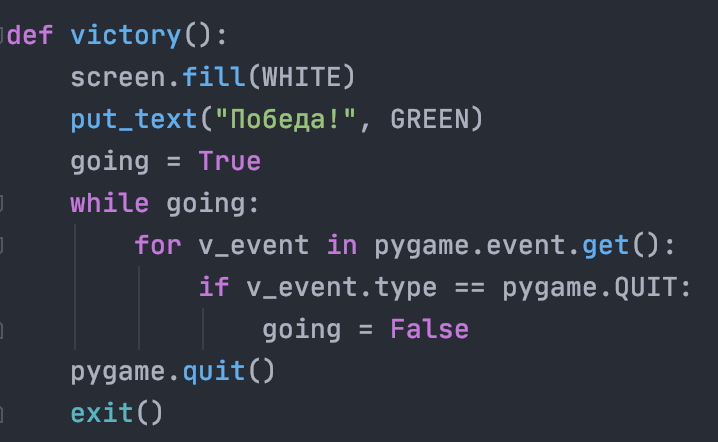
1. Кружок должен появиться снова слева, поэтому присвоим x значение 0.
2. Наша высота снова должна быть рандомно выбрана из списка высот heights.
3. Все ворота снова становятся неактивированными, поэтому gate = -1.
4. Переменная won принимает значение 0 в каждом новом уровне.
5. Добавляем 1 к значению текущего уровня.
6. Определяем скорость кружка в зависимости от значения текущего уровня.

### Победа в игре

Если игрок прошел 10 уровней, мы можем поздравить его с победой и вывести соответствующее сообщение на экран. Добавим условие после цикла for, в котором в случае победы мы обновляем значение уровня, для проверки показателя текущего уровня:

|  |
| --- |
| if lvl > 10:  victory() |

Все верно, для таких целей нам лучше использовать отдельную функцию, в которой будет новый бесконечный цикл лишь с одним условием:



Для начала заполним весь экран белым цветом, а после отобразим сообщение о победе с помощью нашей функции, отвечающей за текст. Создадим новую переменную going, которая изменит свое значение тогда, когда игрок нажмет на крестик для выхода.

# Конец

Поздравляем, вот и готова твоя первая игра. Было не так уж и просто: для такой маленькой игры мы использовали два вида циклов, списки, кортежи, переменные, отвечающие за совершенно разные вещи, функции, которые мы вызываем с разными параметрами. Зато теперь можно выдохнуть и гордо запустить игру, но нельзя расслабляться, ведь дальше нас ждут еще более интересные и сложные игры, которые потребуют максимальной концентрации и отдачи!

Ждем тебя в следующих курсах по играм :)

# Ссылка на проект

Если вдруг что-то не получилось до конца или просто интересно посмотреть реализацию и оформление, прикрепляю ссылку на мой вариант игры. Увидимся!

[Игра про мячик и ворота](https://drive.google.com/file/d/1vvpJbMLEOch0Z6tYHcdJUtZsboddhJwy/view?usp=sharing)