Игра 2: Понг

# 

В этот раз мы создадим классическую игру Pong. Выполним ее в минималистичном черно-белом оформлении с управлением для двух игроков. Пишите код, а после - зовите друга побороться в вашу игру!

# Повторение

## Необходимый софт

* Python. Для написания нашей второй игры мы все также будем использовать Python до версии 3.7 включительно.
* IDE. Так как мы уже познакомились с Pycharm и его функциями, в этот раз вы можете использовать любое удобное вам IDE или текстовый редактор (например, Atom). То, в чем писать код разработчику, решает только сам разработчик. По этой причине Pycharm остается только в качестве рекомендации.

## Импорт, инициализация, экран и цвета

* Импорт.

Для нашей игры мы снова импортируем основную библиотеку Pygame и сразу добавим к ней модуль random:

|  |
| --- |
| import pygame import random |

* Инициализируем модуль pygame:

|  |
| --- |
| pygame.init() |

* Создадим окно игры с нужными нам параметрами:

|  |
| --- |
| SCREEN\_WIDTH = 800 SCREEN\_HEIGHT = 600  screen = pygame.display.set\_mode([SCREEN\_WIDTH, SCREEN\_HEIGHT]) |

* Добавим две переменные для хранения цветов нашей игры:

|  |
| --- |
| WHITE = (255, 255, 255) BLACK = (0, 0, 0) |

## Главный цикл

Вспомним, как мы создавали цикл для того, чтобы игра продолжалась до момента, пока мы ее сами не остановим. Для этого нам понадобится переменная “выключатель”.

|  |
| --- |
| running = 1  while running:  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = 0 |

Тут же добавили проверку event на то, является ли это событие нажатием на крестик. Следует отметить, что данная конструкция непрерывного цикла будет присутствовать в каждой нашей следующей игры, что очень логично для любого рода игр. Ведь внутри них всегда происходит какой-то не заканчивающийся процесс - вечный цикл.

## ФПС

Добавим ограничение кадров в секунду для того, чтобы наша игра работала одинаково на компьютерах с разными процессорами. Для этого используем модуль Clock() внутри time в pygame:

|  |
| --- |
| clock = pygame.time.Clock() running = 1  while running:  for event in pygame.event.get():  if event.type == pygame.QUIT:  running = 0  clock.tick(30) |

Оставим стандартное ограничение в 30 кадров, для нашей игры Понг - в самый раз.

## Итог повторения

Все представленные модули выше мы будем использовать в каждой новой игре. Посмотрите на них еще раз: без каждого модуля не обойтись, чтобы игра функционировала. Возьмем на заметку, что эту часть кода мы почти что будем копировать из игры к игре.

# 

# Цель игры



Сегодня мы будем воссоздавать классическую игру “PONG”. Сама игра является компьютерной версией спортивной игры пинг-понг. Для реализации игры фирма Atari использовала простейшую двумерную графику. Игра появилась в 1972 году и считается, что с ее появлением зародилась индустрия интерактивных развлечений. Сейчас игровая индустрия является одной из крупнейших по прибыли и технологии уже во много раз превосходят старичка Pong. Почему бы не отдать ему дань уважения и не написать свой вариант культовой игры?

В этот раз мы будем писать игру для двух игроков с управлением для каждого - в нашей игре будут две ракетки, по нажатиям клавиш они будут ездить вверх и вниз с целью отбить мячик.

Цель - накопить больше 7 победных очков с условием отрыва на 2 очка от соперника.

# 

# Реализация

## План

Каждый раз начиная любой проект, эффективно будет планировать свои действия. Сначала решить задачку в уме или на листке бумаги, если требуется нарисовать какие-либо логические схемы, а потом - переносить свою идею в код.

Итак, пройдемся по основным пунктам:

* В нашей игре будет два игрока - две ракетки.
* Игроки будут управлять ракетками с помощью клавиш клавиатуры.
* По полю будет летать мяч, который игроки должны отбить.
* Если мяч скроется за пределами поля одного из игроков, другому добавляется одно победное очко.
* Если один игрок достиг показателя в 7 победных очков и оторвался на два очка от соперника, то объявляем о победе этого игрока.

Вот и весь план. Уже появились мысли о том, как это реализовать? На самом деле большей частью программирования является решение задач именно идейно - в голове, ведь написание кода - лишь инструмент для реализации наших идей, а сегодня мы постараемся улучшить наши навыки в управлении этим инструментом. Готовы?

## Пишем код

### Переменные

Давайте пройдемся по уже имеющимся наименованиям:

* running - отвечает за наш главный игровой цикл. Присвоим значение 0 - выйдем из цикла, и игра закончится.
* SCREEN\_WIDTH и SCREEN\_HEIGHT - геометрические параметры нашего окна: ширина и высота, их мы будем использовать для выравнивания отображения фигур.
* screen - объект нашего экрана, окна, на котором мы будем отображать все графические объекты нашей игры.
* WHITE и BLACK - два кортежа для хранения черного и белого цветов в формате RGB кода.
* clock - объект, который хранит в себе информацию о времени работы игры. Используем его для ограничения кадров в секунду.

Добавим новые переменные, пока что только 3:

* left\_player - будем использовать эту переменную для хранения победных очков левого игрока, начальное значение будет равно 0.
* right\_player - храним победные очки правого игрока, начальное значение тоже 0.
* ready - сделаем переменную для хранения готовности игроков начать раунд. Предлагаю запускать каждый раунд по нажатию на пробел, после нажатия ready станет равно 1. Тогда ready перед каждым раундом будет принимать значение 0, присвоим 0 и сейчас.

### Объекты

Давайте определимся, из чего же будет состоять наша игра. Все очень просто: ракетки и мяч. Создадим их в коде:

* Мяч. Создадим его объект с помощью модуля Rect - как мы помним из прошлого урока этот модуль отвечает за прямоугольные объекты в Pygame. Так что технически наш мяч - это квадрат, вы вправе изменить это с вашими знаниями о создании геометрических фигур.

|  |
| --- |
| square = pygame.Rect(SCREEN\_WIDTH // 2 - 10, SCREEN\_HEIGHT // 2 - 10, 20, 20) |

Разберемся что мы подаем в Rect: пойдем с конца. 20 и 20 - это высота и ширина нашего квадрата, выраженная в пикселях. Теперь первые два аргумента: мы хотим, чтобы наш мяч появлялся в центре поля, но вспомним у какой точки определяется координата прямоугольника - это левый верхний угол. Получается, этот угол нам нужно сместить на половину ширины левее и на половину высоты выше, отнимем по 10 от половины ширины экрана и половины высоты экрана, получим мяч идеально в центре.

Добавим мячу начальную скорость:

|  |
| --- |
| velocity\_x = -5 velocity\_y = 0 |

Наш мяч может летать не только по одной прямой, но и по диагонали. Для этого нам понадобится горизонтальная скорость velocity\_x и вертикальная velocity\_y. Velocity - скорость по-английски, всегда удобнее называть переменные по смыслу, которые они несут. Пусть в самом начале мяч летит налево со скоростью 5 пикселей в секунду.

* Ракетки. Предлагаю сделать ракетки по ширине равной нашему квадратному мячу, а по длине - в 4 раза больше.

|  |
| --- |
| left\_pad = pygame.Surface((20, 80)).get\_rect(center=(20, SCREEN\_HEIGHT//2)) right\_pad = pygame.Surface((20, 80)).get\_rect(center=(SCREEN\_WIDTH - 20, SCREEN\_HEIGHT//2)) |

Давайте разберемся в этих двух строчках: left\_pad и right\_pad отвечают за левую и правую ракетку соответственно. Для их создания мы обратились к модулю Surface, который возвращает поверхность с заданными параметрами - 20 пикселей ширина и 80 высота. У поверхности Surface вызвали метод get\_rect, он позволяет задать центр нашей поверхности с помощью специального флага center. Поместим левую ракетку, а именно ее центральную точку, на расстоянии 20 пикселей от левого края и на высоте равной половине высоты экрана. Правую расположим на той же высоте, но чтобы ее центр отодвинуть от правого края, отнимем от значения ширины экрана 20.

Теперь ракетки появятся на одной высоте друг с другом и мячом.

|  |
| --- |
| pads = [left\_pad, right\_pad] |

Создадим лист pads, состоящий из объектов двух ракеток, он понадобится нам позже для эффективного определения удара мяча об ракетки.

### Отрисовка объектов

В конец нашего главного цикла добавим необходимые для отрисовки строчки кода:

|  |
| --- |
| screen.fill(BLACK)  pygame.draw.rect(screen, WHITE, square) pygame.draw.rect(screen, WHITE, right\_pad) pygame.draw.rect(screen, WHITE, left\_pad)  pygame.display.flip()  clock.tick(30) |

Как мы помним, отрисовка происходит таким образом, что мы изменяем координаты определенного объекта, после чего закрашиваем весь экран фоновым цветом: с помощью screen.fill(BLACK) заполним весь экран черным цветом. После чего отрисуем 3 прямоугольных объекта с помощью метода draw.rect() у pygame: первый передаваемый аргумент - это поверхность, на которой будем отображать объект, наш экран; второй аргумент - цвет объекта, выберем белый; третий аргумент - сам объект, в нашем случае это square (наш прямоугольный мяч) и левая и правая ракетки.

После чего обязательно используем метод display.flip() у модуля Pygame для отображения всех изменения на экране.

Добавим эти строчки кода в главный цикл до ограничения кадров в секунду. Модуль clock.tick() удобно вызывать после всех действий в цикле, потому что в таком случае он регулирует то, когда начинать новую итерацию исходя из скорости процессора.

Давайте запустим и проверим, отобразились ли наши объекты на экране. Если все сделали правильно, то сейчас мы увидим 3 минималистических объекта. Отлично!

### Управление ракетками

В прошлой игре мы ловили нажатие на определенную клавишу с помощью event и атрибута KEYDOWN. На этот раз этот вариант оказался неподходящим: дело в том, что этот метод срабатывает при нажатии клавиши и никак не реагирует на то, если клавишу задержали на какое-то время, после чего отжали. Так и будет происходить управление ракетками - мы будем зажимать клавишу вниз или вверх, и все то время, которое мы будем зажимать клавишу, ракетка должна двигаться в сторону выбранного направления. Для этого нам подойдет метод get\_pressed() атрибута key:

|  |
| --- |
| keys = pygame.key.get\_pressed() |

keys сейчас - это список с огромным значением значений-нулей, они принимают значение True - 1, если определенная клавиша зажата в текущей момент.

Теперь мы можем эффективно изменять положение ракеток с помощью созданного списка:

|  |
| --- |
| if keys[pygame.K\_w]:  left\_pad.y -= 8 if keys[pygame.K\_s]:  left\_pad.y += 8  if keys[pygame.K\_UP]:  right\_pad.y -= 8 if keys[pygame.K\_DOWN]:  right\_pad.y += 8 |

Для левой ракетки будем использовать клавиши ‘w’ и ‘s’, а для правой стрелки вверх и вниз. Не забудьте, что клавиши ‘w’ и ‘s’ существуют только на английской раскладке клавиатуры, убедитесь перед запуском игры, что выбрана именно английская раскладка, иначе правая ракетка не будет двигаться. При зажатии клавиши изменяем высоту ракетки - координату y. Напомню, что координата (0, 0) расположена в левом верхнем углу экрана. Давайте добавим эти строчки кода в наш главный цикл до части с прорисовкой фигур и посмотрим, каким образом двигаются наши ракетки.

Все работает прекрасно, но есть небольшая проблема: ракетки могут уехать за край экрана, мы бы хотели этого избежать. Это очень просто сделать, поправим наши последние добавленные строчки таким образом:

|  |
| --- |
| keys = pygame.key.get\_pressed()  if keys[pygame.K\_w] and left\_pad.y > 0:  left\_pad.y -= 8 if keys[pygame.K\_s] and left\_pad.y < SCREEN\_HEIGHT - 80:  left\_pad.y += 8  if keys[pygame.K\_UP] and right\_pad.y > 0:  right\_pad.y -= 8 if keys[pygame.K\_DOWN] and right\_pad.y < SCREEN\_HEIGHT - 80:  right\_pad.y += 8 |

Мы добавили дополнительные условия для изменения координат ракеток. Вспомним, что координата ‘y’ у ракетки - это координата ее левого верхнего угла.

Тогда этот угол при движении ракетки наверх не должен уходить за верхний край экрана, то есть не должен становиться отрицательным - разрешим движение ракеток наверх только при условии, что координата ‘y’ положительная.

Теперь с нижней границей: раз координату ‘y’ мы проверяем у левого верхнего угла, то нижняя часть ракетки находится на расстоянии в 80 пикселей (высота ракетки) от этого угла. Это значит, что ракетка может двигаться, только если координата ‘y’ ее левого верхнего угла меньше чем точка, расположенная на высоте в 80 пикселей от нижней границы экрана (нижняя граница экрана имеет координату равной высоте экрана).

Оставим скорость 8 пикселей в секунду у обеих ракеток, так будет честно.

Давайте запустим и проверим, правильно ли двигаются ракетки.

Сейчас с ракетками все должно быть хорошо.

### Движение мяча

Давайте сначала подумаем о том, как будет происходить движение в теории. У нашего не совсем круглого мяча есть два параметра: горизонтальная скорость по оси ‘x’ и вертикальная скорость по оси ‘y’. Когда мяч ударяется обо что-то он должен менять направление своего движения. На практике для того, чтобы мяч начал движение в обратную сторону, нам всего лишь нужно будет поменять знак у скорости.

Начнем с самого простого момента, но очень важного момента: что же произойдет, когда мяч окажется за пределами поля? Мы должны увеличить значение переменной, которая хранит в себе текущий счет определенного игрока. Давайте добавим это в наш цикл:

|  |
| --- |
| if square.x > SCREEN\_WIDTH or square.x < 0:  if square.x > SCREEN\_WIDTH:  left\_player += 1  else:  right\_player += 1  velocity\_x = random.choice((-5, 5))  velocity\_y = 0  square.x = SCREEN\_WIDTH // 2 - 10  square.y = SCREEN\_HEIGHT // 2 - 10  ready = 0 |

Заходим в условие, если координата нашего квадратного мяча (square) оказалась за правым краем - больше ширины экрана или за левым краем - меньше 0.

Проверяем, если за правым краем - левый игрок получает плюс одно победное очко. Иначе - победное очко получает правый игрок.

После чего реинициализируем основные параметры:

* Скорость мяча по горизонтальной оси станет снова 5 - добавим немного рандома, пусть мяч начинает лететь то вправо, то влево в разные раунды, для этого используем random.choice().
* По вертикальной оси скорость мяча velociti\_y станет равно 0. Изначально мяч летит только по горизонтали.
* Вернем координатам квадратного мяча (square) начальное значение - поместим мяч в центр.
* Сбросим значение переменной ready, для того, чтобы новый раунд начинался по команде игроков.

Само движение мяча реализовать очень просто. Нам нужен метод move\_ip (move in place), этот метод доступен во всех геометрических объектах. move\_ip сдвигает объект на определенные значения относительно текущих координат (эти значения и есть наша скорость). Его альтернатива move - перемещает объект на переданные координаты (подав в move координаты (1, 1), объект окажется на этих координатах), в нашем случае move не подходит. Давайте добавим движение мяча после последнего добавленного блока:

|  |
| --- |
| if ready:  square.move\_ip(velocity\_x, velocity\_y) |

В метод мы подаем изменение координаты (скорость) для каждой из двух осей. Добавим условие, что если мы не нажали пробел, мяч двигаться еще не будет - перед началом раунда у нас будет время настроиться.

Теперь очень важно понять, что же будет происходить, если мы отобьем ракеткой мяч. Все очень просто - мяч отскочит в обратном направлении, но как же это сделать в нашей программе? Для этого нам нужно разобрать такое понятие как “Коллизия”.

### Коллизия

Все очень просто: коллизия - это столкновение объектов. Используется этот термин не только в видеоиграх, но для них он имеет особое значение. Подумайте о разных играх - коллизия используется в каком-либо проявлении используется, наверное, в каждой игре. Даже в нашей прошлой игре мы могли использовать коллизию для определения того, в какие ворота ударился мяч.

Методы для определения столкновения с другими объектами можно вызвать объектов, у которых определена поверхность. Например, у прямоугольного объекта нашего квадратного мяча. Если написать square.collide, то Pycharm подскажет нам доступные методы для определения столкновений:

* collidepoint() - проверяет, если поданная точка находится внутри прямоугольника.
* colliderect() - проверяет, если два прямоугольника столкнулись.
* collidelist() - проверяет, если хотя бы один из прямоугольников в поданном списке столкнулся с исходным.
* collidelistall() - проверяет, если все прямоугольники в списке столкнулись с исходным.
* collidedict() и collidedictall() - работают аналогично методам со списками.

Для нашего случая подойдет вариант, в котором мы будем проверять, столкнулся ли наш мяч хотя бы с одним из объектов в поданном списке. Помните, мы создали список pads, содержащий в себе две ракетки, до этого? Его мы и подадим как аргумент в метод collidelist().

Предлагаю такое решение для отбивания мяча ракетками:

|  |
| --- |
| if square.collidelist(pads) >= 0:  velocity\_x = -velocity\_x \* 1.1  velocity\_y = random.randint(-8, 8) |

Очень просто и понятно: метод collidelist() вернет кол-во столкновений с нашим square. Нам нужно хотя бы одно. Заходим в условие, меняем знак у скорости (так мяч полетит в обратное направление) и умножаем на 1.1 - это задаст темп игре: чем больше раз отбили мяч, тем быстрее он будет двигаться. После столкновения с ракеткой, мяч полетит с обратной и немного увеличенной горизонтальной скоростью, но что делать с вертикальной? Все просто: возьмем рандомное значение из интервала (-8, 8), никакой разница, как оказалось, как именно мы отобьем мяч нет - мяч полетит вверх или вниз, исходя только лишь из рандома.

Добавьте эти строчки кода после блока, который описывает движение мяча. Давайте запустим и проверим, можем ли мы отбивать мяч ракетками. Все должно заработать.

Кроме одной проблемы: мяч вылетает за верхний и нижние края экрана.

### Столкновение с краями

Для нашего мяча осталось лишь одно ограничение: как только он коснется верхнего или нижнего края, он должен отскочить и полететь в другую сторону относительно вертикальной оси. Реализуем это в коде:

|  |
| --- |
| if square.y >= SCREEN\_HEIGHT - square.height or square.y <= 0:  velocity\_y = -velocity\_y |

Меняем направление вертикальной скорости мяча, если он ударился об нижний край (его левый верхний угол находится ниже координаты SCREEN\_HEIGHT - square.height высота квадратика), и если он ударился об верхний край (левый верхний угол оказался выше отметки 0).

Добавим эту проверку после проверки на столкновение с ракетками.

Запустите и проверьте, сейчас наша игра с точки зрения механики полностью готова.

### Готовность игроков

Как мы и обсуждали, наш мяч начнет двигаться, только когда игроки будут готовы. Для этого используем нажатие клавиши пробел. Добавим проверку в наш цикл считывания событий в основном цикле:

|  |
| --- |
| for event in pygame.event.get():  if event.type == pygame.QUIT:  running = 0  if not ready and event.type == pygame.KEYDOWN:  if event.key == pygame.K\_SPACE:  ready = 1 |

Заходим в условие, если ready не истина (равно 0), и если нажата клавиша. Если нажат пробел, меняем значение переменной ready на 1.

### Условие победы

Условием нашей победы будет набрать 7 победных очков и оторваться от соперника на 2 или больше очка, добавим эту проверку в главный цикл после цикла с событиями:

|  |
| --- |
| if left\_player >= 7 or right\_player >= 7 and abs(left\_player - right\_player)>1:  victory() |

Правая часть условия вызывает метод abs() - взять по модулю. Нам нужно чтобы разница очков была больше 1, вычтем из кол-ва победных очков левого игрока кол-во победных очков правого игрока, возьмем модуль от полученного числа, и получим положительную разницу победных очков.

Как вы могли заметить, мы вызываем функцию victory(), если кто-то из игроков победил. В этой функции всего лишь объявим о победе одного из игроков.

### Функция victory()

|  |
| --- |
| def victory():  player = '1' if left\_player > right\_player else '2'  screen.fill(BLACK)  put\_text(("Player " + player + " won!"), SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2)  pygame.display.flip()  while True:  for ev in pygame.event.get():  if ev.type == pygame.QUIT:  pygame.quit()  exit() |

* Переменная player будет хранить в себе строку с номером игрока. Для того, чтобы написать присваивание в одну строку, используем тернарный оператор.
* Заполним экран черным цветом.
* Вызовем функцию put\_text, подадим в нее конкатенированную строку и координаты центра экрана. За основу функции put\_text мы возьмем аналогичную функцию из прошлой игры.
* Обновим экран с помощью flip().
* Воспроизведем вечный цикл с одной лишь опцией - выйти из игры нажатием на крестик. Игра на этот момент закончена.

### 

### Функция put\_text()

Модифицируем эту же функцию из игры №1. Получим:

|  |
| --- |
| font = pygame.font.Font(None, 36)   def put\_text(message, x, y):  text = font.render(message, True, WHITE)  place = text.get\_rect(center=(x, y))  screen.blit(text, place) |

* До функции определим объект font, содержащий в себе информацию о шрифте.
* Функция будет принимать строку - message и координаты центра прямоугольной области, в которой будет находиться текст.
* Определим объект text с помощью метода render(), передадим туда флаг True для сглаживания текста и цвет WHITE для белого цвета букв.
* Теперь мы хотим поместить наш текст в центр, зададим прямоугольную область у нашего объекта text с флагом center и переданными координатами для него.
* Отобразим изменения на экране с помощью blit().

### Отображение счета

Добавим вызов функции put\_text для отображения счета игроков в главном цикле после того, как заполнили экран черным цветом:

|  |
| --- |
| put\_text(("Player 1 score: " + str(left\_player)), SCREEN\_WIDTH // 2 - 200, 40) put\_text(("Player 2 score: " + str(right\_player)), SCREEN\_WIDTH // 2 + 200, 40) |

Передадим сконкатинированную строку для каждого игрока и координаты, удаленные на 200 пикселей от центра и 40 от верхнего края экрана.

# 

# Заключение

Поздравляю с написанием второй игры! Отличная работа, и теперь можно позвать друга или родственника и проверить, кто сильнее в Pong!

Благодаря этой игре мы изучили такие важнейшее понятие как коллизия и потренировались с применением ее. Разработали план и логику игры, после чего реализовали ее в коде. На это раз мы обошлись парой функций и несколькими переменными, но уже в следующей игре мы будем знакомиться с применением классов в играх - а это еще одна неотъемлемая деталь разработки игр. Не забывайте, что вы можете редактировать игру на ваше усмотрение, и я только приветствую такую практику!

# Ссылка на проект

Для того, чтобы увидеть мой вариант решения, переходите по прикрепленной ссылке:

<https://drive.google.com/file/d/1lphzpXHJeWd_Zr3VO0HhDmOL0b7A-igq/view?usp=sharing>