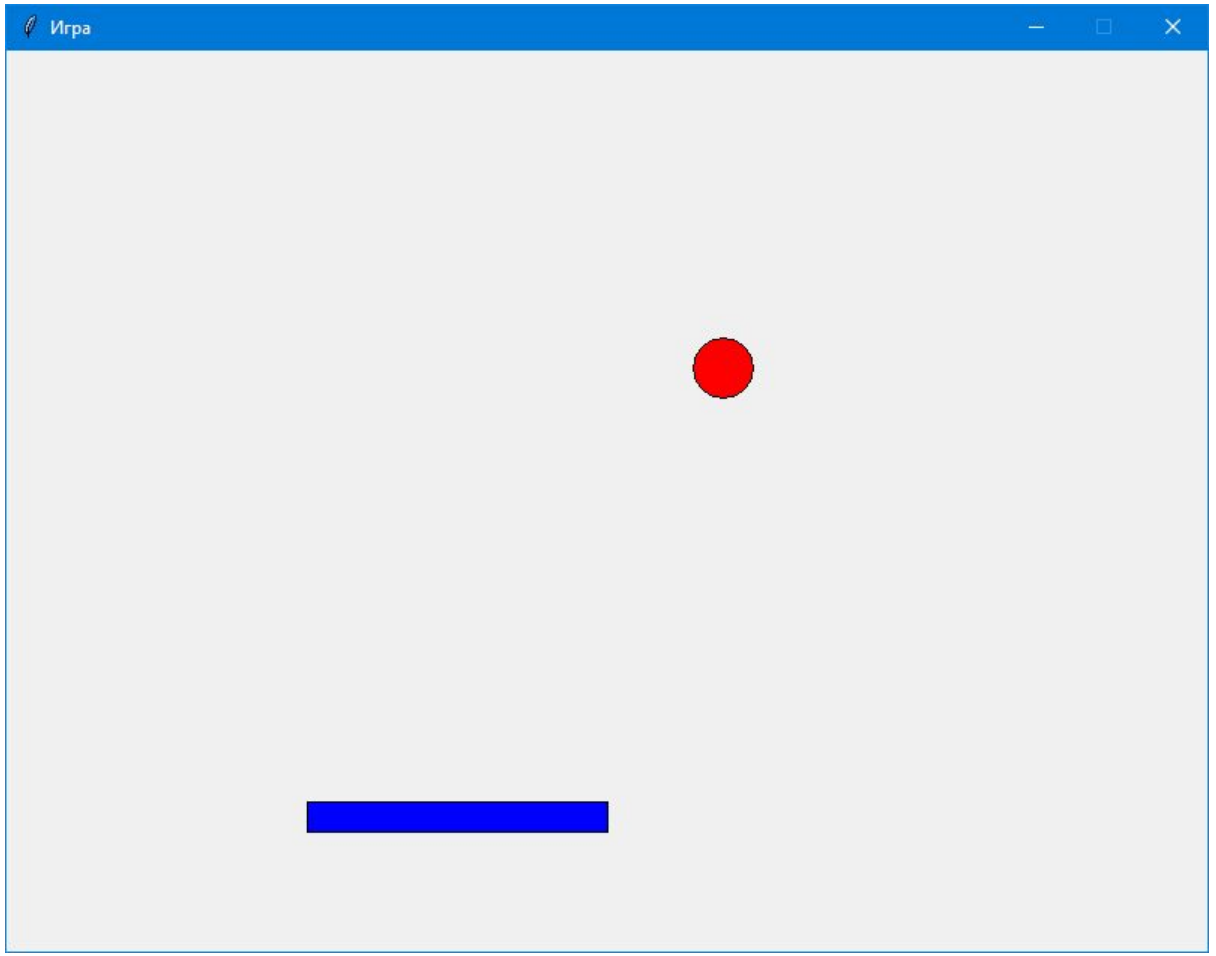
Прыгающий мяч



Создаем игровой холст

```
from tkinter import *
import random
import time
tk = Tk()
tk.title("Игра")
tk.resizable(0, 0)
tk.wm_attributes("-topmost", 1)
canvas = Canvas(tk, width=500, height=400, bd=0,
highlightthickness=0)
canvas.pack()

while True:
    tk.update()
```

Создаем класс для мяча

- создать класс под названием Ball, принимающий в качестве аргументов функции __init__ холст и цвет мяча;
- сохранить в свойстве объекта холст, чтобы в дальнейшем рисовать на нем мяч;
- Изобразить на холсте круг, заполненный переданным в аргументе цветом;
- сохранить идентификатор, который вернет нам функция рисования круга, поскольку с его помощью мы будем перемещать мяч по экрану;
- переместить нарисованный круг в центр холста.

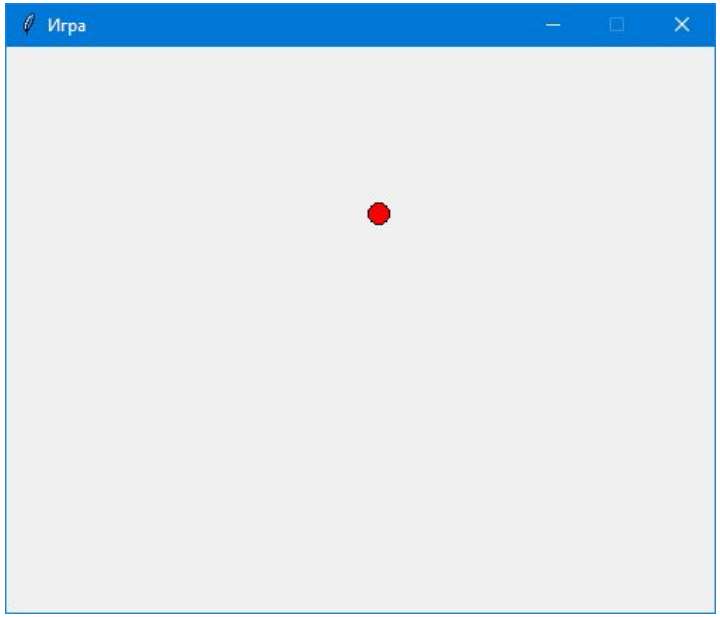
Этот текст после import time

```
class Ball:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_oval(10,10,25,25, fill=color)
        self.canvas.move(self.id, 245,100)

    def draw(self):
        pass
```

Итак, у нас есть класс Ball, и теперь нужно создать объект этого класса

```
23  ball = Ball(canvas, 'red')
24
25  while True:
26      tk.update_idletasks()
27      tk.update()
28      time.sleep(0.01)
29
```



Добавим движение

Перемещение мяча

Изменим функцию draw()

```
def draw(self):
    self.canvas.move(self.id, 0, -1)
```

В главный цикл игры нужно внести еще одно изменение. Добавим
в тело цикла `while` (это и есть главный цикл) вызов функции объекта-мяча `draw`. Вот так:

```
25  while True:
26      ball.draw()
27      tk.update_idletasks()
28      tk.update()
29      time.sleep(0.01)
30
```

Отскоки мяча

Добавим несколько свойств в функцию __init__

```
def __init__(self, canvas, color):

    self.canvas = canvas
    self.id = canvas.create_oval(10, 10, 25, 25, fill=color)
    self.canvas.move(self.id, 245, 100)
    starts = [-3, -2, -1, 1, 2, 3]
    random.shuffle(starts)
    self.x = starts[0]
    self.y = -3
    self.canvas_height = self.canvas.winfo_height()
    self.canvas_width = self.canvas.winfo_width()
```

Отскоки мяча

- Добавим несколько свойств в функцию draw

```
def draw(self):
    self.canvas.move(self.id, self.x, self.y)
    pos = self.canvas.coords(self.id)
    if pos[1] <= 0:
        self.y = 3
    if pos[3] >= self.canvas_height:
        self.y = -3
    if pos[0] <= 0:
        self.x = 3
    if pos[2] >= self.canvas_width:
        self.x = -3
```

Создаем ракетку

Создадим новый класс для ракетки

```
class Paddle:
    def __init__(self, canvas, color):
        self.canvas = canvas
        self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
        self.canvas.move(self.id, 200, 300)

    def draw(self):
        pass
```

Управление ракеткой

Добавим две новые функции

```
    def turn_left(self, evt):
        self.x = -2

    def turn_right(self, evt):
        self.x = 2
```

Добавим в главную функцию новые свойства

```
def __init__(self, canvas, color):
    self.canvas = canvas
    self.id = canvas.create_rectangle(0, 0, 100, 10, fill=color)
    self.canvas.move(self.id, 200, 300)
    self.x = 0
    self.canvas_width = self.canvas.winfo_width()
    self.canvas.bind_all('<KeyPress-Left>', self.turn_left)
    self.canvas.bind_all('<KeyPress-Right>', self.turn_right)
```

И подготовим функцию draw

```
def draw(self):
    self.canvas.move(self.id, self.x, 0)
    pos = self.canvas.coords(self.id)
    if pos[0] <= 0:
        self.x = 0
    elif pos[2] >= self.canvas_width:
        self.x = 0
```

Проверка на столкновение мяча с ракеткой

Добавим в функцию __init__ класса Ball еще один аргумент —объект-ракетку:

```
    def __init__(self, canvas, color, paddle):

        self.canvas = canvas
        self.paddle = paddle
```

Теперь нужно изменить код создания объекта-мяча с учетом нового аргумента — ракетки. Этот код находится в конце программы перед
главным циклом:

```
paddle = Paddle(canvas, 'blue')
ball = Ball(canvas,paddle, 'red')
while 1:
    ball.draw()
    paddle.draw()
```

Столкновение ракетки с мячом

- Создадим в классе ball новую функцию

```
def hit_paddle(self, pos):
    paddle_pos = self.canvas.coords(self.paddle.id)
    if pos[2] >= paddle_pos[0] and pos[0] <= paddle_pos[2]:
        if pos[3] >= paddle_pos[1] and pos[3] <= paddle_pos[3]:
            return True
    return False
```

Изменим функцию draw

```
def draw(self):
    self.canvas.move(self.id, self.x, self.y)
    pos = self.canvas.coords(self.id)
    if pos[1] <= 0:
        self.y = 3
    if pos[3] >= self.canvas_height:
        self.y = -3
    if self.hit_paddle(pos) == True:
        self.y = -3
    if pos[0] <= 0:
        self.x = 3
        if pos[2] >= self.canvas_width:
            self.x = 3
    if pos[2] >= self.canvas_width:
        self.x = -3
```

Добавим проигрыш

Сперва создадим в теле функции __init__ класса Ball свойство hit_bottom (признак того, что мяч достиг нижней границы холста). Добавим этот код в самый низ функции __init__:

```
self.canvas_height = self.canvas.winfo_height()
self.canvas_width = self.canvas.winfo_width()
self.hit_bottom = False
```

Главный цикл соответственно изменим на:

```
while 1:
    if ball.hit_bottom == False:
        ball.draw()
        paddle.draw()
    tk.update_idletasks()
    tk.update()
    time.sleep(0.01)
```

Изменим обработку столкновения мяча с нижней границе экрана, чтобы игра останавливалась

```
def draw(self):
    self.canvas.move(self.id, self.x, self.y)
    pos = self.canvas.coords(self.id)
    if pos[1] <= 0:
        self.y = 3
    if pos[3] >= self.canvas_height:
        self.hit_bottom=True
    if self.hit_paddle(pos) == True:
        self.y = -3
```