Применение tkinter для разработки игры Arkanoid.

Источники информации:
1-ая часть: https://rutube.ru/video/f2fa039b53ac62f30c3fd5d3a8c7ae45/
2-ая часть: https://www.youtube.com/watch?v=vSuV32y2xGQ

В предлагаемом к изучению материале рассматривается разработка всем известной игры Arkanoid, в качестве инструмента – tkinter.

Занятие второе практическое, некоторые теоретические вопросы рассматриваются в ходе написания кода игры.

В этом занятии будем заниматься совершенствованием кода игры, сразу отметим то, к чему будем стремиться:

- В верхней части окна игры размещаются цветные блоки, которые мячом нужно выбить;
- За уничтожение блоков начисляются баллы;
- Размер игрового поля будет увеличен по ширине;
- Счётчик теперь будет расположен посередине игровой платформы и перемещается вместе с ней;
- Как и ранее мяч в игре должен отскакивать от стенок и от платформы, а когда коснётся нижней границы – игра завершается с проигрышем.

Файл игры назовём **arkanoid2.py**, чтобы он отличался от уже написанного ранее.

Обсудим изменения кода программы.
Кроме импорта **tkinter**, импортируем **random** – он нам будет нужен для создания разноцветных блоков.
Ширину холста, а значит игрового пространства увеличим до 1200 пикселей.

```
import tkinter
import random

WIDTH, HEIGHT = 1200, 500  # ширина и высота холста
BALL_RADIUS = 20  # радиус, которым будет рисоваться мяч
```

```
TIMEOUT = 30  # интервал между кадрами игры, в миллисекундах

PLATFORM_H = 50
PLATFORM_W = 250
x1 = WIDTH // 2  # начальная позиция центра платформы по оси x

x, y = WIDTH // 2, HEIGHT // 2  # координаты центра экрана
vx, vy = -10, -10  # перемещение по x, y за один интервал времени
```

Рассмотрим, что нам нужно сделать чтоб в программе появились цветные кирпичи:

Зададим часть по высоте игрового поля, которая будет занята кирпичами, а также определим сколько рядов и колонок кирпичей:

```
BRICKS_PART = 0.3
ROWS, COLS = 3, 10
```

Далее часть кода из предыдущей программы:

```
points = 0
game_mode = True

root = tkinter.Tk()

canvas = tkinter.Canvas(root, width=WIDTH, height=HEIGHT)
canvas.pack()

ball = canvas.create_oval(
    x - BALL_RADIUS, y - BALL_RADIUS, x + BALL_RADIUS, y + BALL_RADIUS,
    fill='yellow')

platform = canvas.create_rectangle(
    x1 - PLATFORM_W // 2, HEIGHT,
    x1 + PLATFORM_W // 2, HEIGHT - PLATFORM_H, fill='green')
```

Так как количество очков хотим отображать на платформе, то изменим метку для вывода очков:

```
score = canvas.create_text(
    x1, HEIGHT - PLATFORM_H // 2, text='0',
    font=('Courier', 32, 'bold'), fill='white')
```

Далее рассчитываем размеры одного кирпича и создаём пустой список для создаваемых разноцветных кирпичей:

```
brick_w, brick_h = WIDTH / COLS, HEIGHT * BRICKS_PART / ROWS
bricks = []
```

Для заполнения списка выполним:

```
for row in range(ROWS):
    for col in range(COLS):
        xb, yb = col * brick_w, row * brick_h
        red, green, blue = (random.randint(0, 255) for _ in range(3))
        color = f'#{red:0>2x}{green:0>2x}{blue:0>2x}'
        bricks.append(canvas.create_rectangle(
            xb, yb, xb + brick_w, yb + brick_h, fill=color))
```

```
# Функция хода игры, т.е. программируем один кадр игры
def game():
    global x, y, vx, vy, points, game_mode

    # Изменим координаты для перемещения мяча в кадре
    x, y = x + vx, y + vy
    canvas.coords(  # этим методом холста перемещаем мяч
        ball, x - BALL_RADIUS, y - BALL_RADIUS,
        x + BALL_RADIUS, y + BALL_RADIUS)

    if y <= BALL_RADIUS:  # если мяч коснулся верхней границы экрана
        vy = abs(vy)  # движение мяча в положительном направлении

    # Похожим образом обрабатываем прикосновение мяча к краям по оси x
    if x <= BALL_RADIUS or x >= WIDTH - BALL_RADIUS:
        vx = -vx

    # Опишем отбив мяча платформой
    if x1 - PLATFORM_W // 2 <= x <= x1 + PLATFORM_W // 2 and \
       y == HEIGHT - (BALL_RADIUS + PLATFORM_H):
        vy = - vy
        points += 1
        canvas.itemconfig(score, text=str(points))
```

Здесь опишем удаление разбитого мячом кирпича:

```
    brick = get_brick()  # получаем разбитый кирпич
    if brick:
        vy = - vy  # Меняем направление движения мяча

        # Удаляем этот кирпич
        canvas.delete(brick)
        bricks.pop(bricks.index(brick))
        points += 1  # Увеличим очки за разбитый кирпич
        canvas.itemconfig(score, text=str(points))

    root.update()
    if y < (HEIGHT - BALL_RADIUS):
        root.after(TIMEOUT, game)
    else:
        game_mode = False
        canvas.create_text(WIDTH // 2, HEIGHT // 2, text='GAME OVER',
                           fill='red', font=(None, 50))

def keyboard(event):
    global x1
    if event.keycode == 37:
        x1 -= 50
    if event.keycode == 39:
        x1 += 50
    if game_mode:
        canvas.coords(platform, x1 - PLATFORM_W // 2, HEIGHT,
                      x1 + PLATFORM_W // 2, HEIGHT - PLATFORM_H)
        canvas.coords(score, x1, HEIGHT - PLATFORM_H // 2)
```

```
def mouse_move(event):
    global x1
    x1 = event.x
    if game_mode:
        canvas.coords(platform, x1 - PLATFORM_W // 2, HEIGHT,
                      x1 + PLATFORM_W // 2, HEIGHT - PLATFORM_H)
        canvas.coords(score, x1, HEIGHT - PLATFORM_H // 2)
```

Далее мы опишем функцию, которая определяет какой кирпич нужно разбить:

```
def get_brick():
    # В цикле перебираем все кирпичи
    for brick in bricks:
        # Извлекаем координаты каждого кирпича
        #     xb1, yb1 – левый верхний угол
        #     xb2, yb2 – правый нижний угол
        xb1, yb1, xb2, yb2 = canvas.coords(brick)
        # Если центр мяча находится внутри пары координат кирпича
        # по горизонтали и вертикали, тогда - возвращаем этот кирпич
        if xb1 < x < xb2 and yb1 < y < yb2:
            return brick

game()
root.bind('<Key>', keyboard)
canvas.bind('<Motion>', mouse_move)
root.mainloop()
```

После запуска программы мы видим работоспособную программу:

- мяч движется, отражается от стенок и от платформы до тех пор, пока не коснётся нижней границы окна;
- количество выбитых очков отображается на самой платформе.

На самом деле в нашей программе имеется серьёзный недостаток – нарушен принцип программирования DRY «не повторяйся». В двух функциях: **keyboard()** и **mouse_move()** имеется абсолютно одинаковые строки кода:

```
if game_mode:
        canvas.coords(platform, x1 - PLATFORM_W // 2, HEIGHT,
                      x1 + PLATFORM_W // 2, HEIGHT - PLATFORM_H)
        canvas.coords(score, x1, HEIGHT - PLATFORM_H // 2)
```

Эти строки смещают платформу и отображают счёт на ней. Имеет смысл для выполнения этих действий реализовать отдельную функцию:

```
def move_platform_and_score():
    if game_mode:
        canvas.coords(platform, x1 - PLATFORM_W // 2, HEIGHT,
                      x1 + PLATFORM_W // 2, HEIGHT - PLATFORM_H)
        canvas.coords(score, x1, HEIGHT - PLATFORM_H // 2)
```

Теперь, повторяющиеся строки кода удалим из функций, а вместо них сделаем вызов только что написанной функции:

```
move_platform_and_score()
```

Как результат, мы выполнили правило DRY в нашей программе.

Проверим получившуюся программу на работоспособность и убедимся, что всё работает.

На этом занятие завершено.

Есть ещё одна любопытная реализация этой игры в гитхабе по ссылке:
https://github.com/Solovman/Arkanoid