Урок tkinter

Практика по работе с графикой и tkinter

Аннотация

В этом уроке мы научимся писать небольшие программы, которые работают не с консолью, а с окном приложения (графическим пользовательским интерфейсом, GUI). Для этого нам понадобится дополнительная функциональность — библиотека tkinter. То, чем мы будем заниматься — только самое начало программирования GUI, поэтому не стоит ожидать того, что мы создадим Photoshop. Но все впереди, и когда-нибудь мы сделаем и это.

1. Программирование в графическом интерфейсе

Согласитесь, что скучно все время видеть перед собой черный экран, который мы называем консолью. На нем нельзя рисовать, пользоваться «мышкой» или джойстиком, организовывать привычный нам оконный интерфейс. Сегодня и на нескольких следующих занятиях мы познакомимся с функциональностью языка Python, которая позволит нам все же немного поработать в графическом режиме.

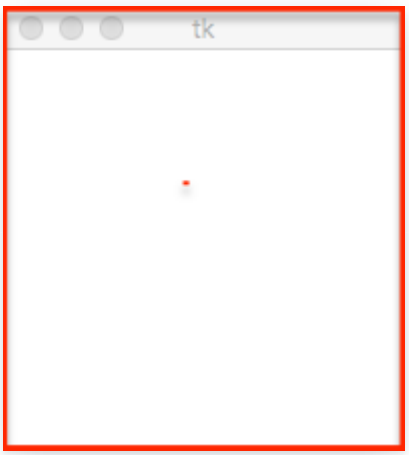
Как вы уже догадываетесь, не вся функциональность языка Python включена в, что называется, стандартную поставку. Какие-то вещи надо добавлять к этому коробочному решению. Эти «добавки» называются библиотеками. Скоро вы узнаете о них подробнее, а пока мы подключим лишь одну из них, которая называется tkinter, и которая поможет нам поработать с графикой.

Попробуйте написать и выполнить следующий код:

```
import tkinter
```

```
master = tkinter.Tk()
master.mainloop()
```

У вас должно открыться пустое окошко, как на рисунке:



Если при запуске Python ответит `ImportError: No module named 'tkinter'`, то это означает, что библиотека tkinter на вашем компьютере отсутствует и нужно ее установить.

Установка tkinter

Установить стандартный пакет библиотек в Python очень просто, для этого есть специальный менеджер пакетов — pip. Чтобы воспользоваться им, запустите терминал (в Windows для этого нужно нажать Пуск, набрать cmd и нажать Enter). В терминале наберите:

```
pip install tkinter
```

И следуйте указаниям. Обычно надо со всем согласиться.

Но вернемся к нашему коду и поймем, что же делают написанные в нем строки.

`import tkinter` — просто подключение пакета, с такими строчками мы уже встречались.

Вторая строчка — `master = tkinter.Tk()` — создает основное окно и дает ему имя master. Дальше мы будем наполнять его содержимым.

Третья строчка — `master.mainloop()` — самая необычная. Дело в том, что приложение с графическим интерфейсом работает не так, как консольные приложения, которые мы писали раньше. В консольных приложениях компьютер исполнял наши инструкции одну за другой. В программе с графическим интерфейсом такой подход работать не будет, потому что компьютер должен реагировать на действия пользователя. Поэтому вместо того, чтобы выполнять команды, он ждет сигналов от пользователя, которые нужно как-то обрабатывать.

Основной цикл

Команда `master.mainloop()` запускает для нашего окна **основной цикл**. Он получает информацию о важных событиях (нажатии клавиш, движениях мышки и пр.), обрабатывает эти события и перерисовывает окно.

Так работают все приложения с графическим интерфейсом. Если вы пользуетесь ОС Windows, то наверняка встречали сообщение, что какая-то программа не отвечает. Обычно это означает, что внутри основного цикла что-то пошло не так (например, программа вошла в бесконечный цикл или зависла). Система продолжает посылать программе сообщения о событиях, но та перестала их принимать, потому что занята чем-то другим.

Обратите внимание, что выполнение программы как бы приостанавливается, когда она доходит до команды `master.mainloop()`, и продолжается только после закрытия окна. Попробуйте предсказать, что произойдет со следующей программой (а потом проверьте свое предположение):

```
import tkinter

master = tkinter.Tk()
master.mainloop()
x = input()
print(x)
```

2. Учимся рисовать

Прежде чем писать программу, по-настоящему взаимодействующую с пользователем, мы научимся рисовать.

Создание холста

Для начала нужно создать холст (canvas), на котором потом будут располагаться картинки. Это действие делает строка:

```
canvas = tkinter.Canvas(master, bg='blue', height=300, width=600)
```

```
import tkinter

master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='blue', height=300, width=600)
canvas.pack()
master.mainloop()
```

Попробуйте поменять параметры `bg` (сокращение для background), `height` и `width`. Цвет может быть либо фиксированным названием цвета (white, black, red, green, blue, cyan, yellow, magenta) либо строкой вида «#RRGGBB». RR, GG и BB — значения красной, синей и зеленой компоненты в диапазоне от 0 до 255, записанные в шестнадцатеричной системе счисления.

Поэкспериментируйте с этим.

Добавление холста в окно

Команда `canvas.pack()` добавляет в окно `master` созданный нами холст — `canvas`. То же самое

можно сделать командой `canvas.grid()`, но работают они немного по-разному:

- `grid()` располагает объекты в ячейках виртуальной сетки, которую мы накладываем на наше окно
- `pack()` же пытается их разместить (упаковать) самостоятельно

Посмотрите, что меняется, если заменить `canvas.pack()` на `canvas.grid()`.

Тренировочная: напишите функцию, которая принимает на вход три числа (красную, зеленую и синюю компоненту) и выдает цвет в описанном выше формате.

Ну давайте уже наконец что-то нарисуем:

```
import tkinter

master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='white', height=600, width=600)
canvas.create_line((0, 0), (600, 600), fill='red')
canvas.pack()
master.mainloop()
```

Рисование линии

Мы нарисовали линию на холсте, передав команде `create_line` сначала две пары — координаты точек, а затем параметр `fill` — цвет линии:

```
canvas.create_line((0, 0), (600, 600), fill='red')
```

Команде `create_line` можно передать больше точек — тогда получится ломаная:

```
import tkinter

master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='white', height=600, width=600)
canvas.create_line((0, 0), (300, 200), (600, 600),
                   (200, 300), (0, 0), fill='red')
canvas.pack()
master.mainloop()
```

Теперь научимся рисовать круг или эллипс:

```
import tkinter

master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='white', height=600, width=600)
canvas.create_oval((0, 0), (100, 100), fill='red')
canvas.create_oval((200, 200), (300, 400), fill='green')
canvas.pack()
```

```
master.mainloop()
```

Если нужно нарисовать много эллипсов, можно использовать цикл:

```
import tkinter

master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='white', height=600, width=600)
for x in range(0, 600, 20):
    canvas.create_oval((x, x), (x + 20, x + 20), fill='red')
canvas.pack()
master.mainloop()
```

В качестве практики найдите в интернете, какие еще команды рисования есть у холста (tkinter.Canvas) и поэкспериментируйте с ними.

3. Взаимодействие с пользователем

Теперь пришло время научить нашу программу взаимодействовать с пользователем. Для этого нужно связать каждое его действие с функцией, которая будет реагировать на действия. Это делается с помощью команды `bind`.

Эта команда получает в качестве входных данных название произошедшего события и функцию, которую нужно выполнить, когда событие происходит:

```
import tkinter


def draw(event):
    canvas.create_oval((100, 100), (300, 300), fill='red')


master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='blue', height=600, width=600)
canvas.pack()
master.bind("<KeyPress>", draw)
master.mainloop()
```

В приведенном примере мы обрабатываем событие KeyPress — нажатие любой клавиши. Tkinter знает много разных событий, вы можете самостоятельно прочитать про них в документации. Пока же прочие события нам не понадобятся.

Важно!

Команда `master.bind("", draw)` означает, что каждый раз, когда пользователь нажимает клавишу, вызывается функция `draw`. В качестве аргумента этой функции передается переменная `event`, в которой содержится описание события.

(Вредный совет: теперь вы тоже можете сделать так, чтобы ваша программа зависла и не отвечала. Для этого нужно добавить в обработку события бесконечный цикл.)

Обратите внимание, что функция получает на вход **один** аргумент — **описание события**, которое ее вызывает. В этой переменной хранится вся возможная информация о событии.

В частности, можно узнать, какая клавиша была нажата, и в зависимости от этого закрашивать круг нужным цветом:

```
import tkinter


def draw(event):
    if event.char == 'r':
        canvas.create_oval((100, 100), (300, 300), fill='red')
    if event.char == 'g':
        canvas.create_oval((100, 100), (300, 300), fill='green')
    if event.char == 'b':
        canvas.create_oval((100, 100), (300, 300), fill='blue')


master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='blue', height=600, width=600)
canvas.pack()
master.bind("<KeyPress>", draw)
master.mainloop()
```

Нажатие клавиши

На самом деле, информация о том, какая клавиша нажата, хранится сразу в нескольких полях («составляющих» переменной `event`):

- `char` — символ, который появляется при нажатии клавиши. Однако это поле определено не всегда: например, когда вы нажимаете Shift, никакого символа не появляется
- `keysym` — символическое описание нажатой клавиши (например, Return, Shift_L (левый шифт), Alt_R и т.п.)
- `keysym_num` — число, соответствующее этому описанию
- `keycode` — код нажатой клавиши. Клавиши могут иметь одинаковый код, но разный keysym, мы встретимся с таким случаем чуть дальше

Как узнать keysym нужной вам клавиши? Можно посмотреть в интернете, но интереснее написать программу, которая их выводит.

Для этого нам понадобится новый инструмент — **поле с текстом**. Этот инструмент называется `Label`:

```
import tkinter
```

```
master = tkinter.Tk()
label = tkinter.Label(master, text="Hello world!")
label.pack()
master.mainloop()
```

Label

Для того чтобы поменять текст на уже существующем `Label`, надо воспользоваться функцией `config`, передав ей параметр `text`:

```
label.config(text="Новый текст")
```

Давайте напишем программу, которая выводит значение keysym для нажатой клавиши:

```
import tkinter


def show_key(event):
    label.config(text=event.keysym)


master = tkinter.Tk()
label = tkinter.Label(master, text="Hello world!")
label.pack()
master.bind("<KeyPress>", show_key)
master.mainloop()
```

Виджеты

Label — еще один инструмент отображения данных в tkinter. В общем такие инструменты принято называть **виджетами** (widgets). Их можно комбинировать в одном окне.

```
import tkinter


def key_pressed(event):
    label.config(text=event.keysym)
    canvas.create_oval((100, 100), (300, 300), fill='green')


master = tkinter.Tk()
label = tkinter.Label(master, text="Hello world!")
label.pack()
canvas = tkinter.Canvas(master, bg='blue', height=600, width=600)
canvas.pack()
```

```
master.bind("<KeyPress>", key_pressed)
master.mainloop()
```

4. Движение объектов

Теперь напишем простую игру. Для начала давайте научимся **двигать** объекты.

Важно!

Важная особенность холста (Canvas) в tkinter заключается в том, что нарисованный объект всегда остается монолитным — с ним можно выполнять разные операции, не затрагивая при этом другие объекты.

Все функции вида `canvas.create_[что угодно]` возвращают целое число — **идентификатор** объекта на холсте. Зная это число, можно, например, двигать объект.

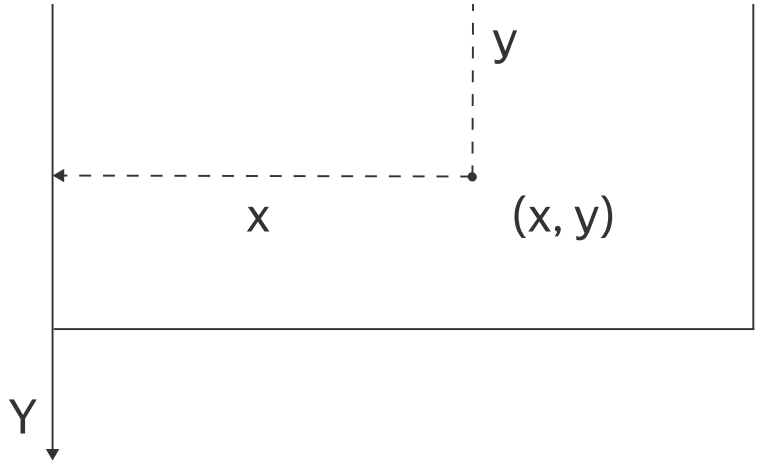
В следующем примере при нажатии на любую клавишу кружочек сдвинет вправо и вниз на 10 позиций — пикселей.

```
import tkinter


def key_pressed(event):
    canvas.move(oval, 10, 10)


master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='blue', height=600, width=600)
oval = canvas.create_oval((300, 300), (310, 310), fill='red')
canvas.pack()
master.bind("<KeyPress>", key_pressed)
master.mainloop()
```

Движение объекта

Чтобы передвинуть объект, нужно передать функции `canvas.move` идентификатор объекта и два числа: изменение координаты x и изменение координаты y.

Обратите внимание на особенность координатной сетки экрана монитора: начало координат располагается в верхнем левом углу экрана. Ось X направлена слева направо, ось Y — сверху вниз.



Важно!

Объекты можно не только двигать, но и произвольно менять их положения и свойства. Для этого нам помогут следующие функции:

- `canvas.coords(object)` — узнать координаты объекта (для овала это будет четверка координат, для линии — все координаты, с которыми она создавалась)
- `canvas.coords(object, new_coordinates)` — задать объекту новые координаты
- `canvas.itemconfig(object, ...)` — задать любые свойства объекта. Например, canvas.itemconfig(object, fill='red') — изменить цвет объекта на красный

Например:

```
import tkinter


def key_pressed(event):
    if event.keysym == 'space':
        canvas.coords(oval, (300, 300, 310, 310))
    if event.keysym == 'Up':
        canvas.move(oval, 0, -10)
    if canvas.coords(oval)[1] < 50:
        canvas.itemconfig(oval, fill='red')


master = tkinter.Tk()
canvas = tkinter.Canvas(master, bg='blue', height=600, width=600)
oval = canvas.create_oval((300, 300), (310, 310), fill='green')
canvas.pack()
master.bind("<KeyPress>", key_pressed)
master.mainloop()
```

5. Приступаем к игре

Теперь приступим к игре. Игрок в ней ходит по сетке размером N_X на N_Y с шагом step. Ему нужно добраться до выхода, причем начальное положение игрока и выхода определяется случайно.

Шаблон программы у нас уже есть:

```
import tkinter
import random


def move_wrap(obj, move):
    canvas.move(obj, move[0], move[1])
    # Здесь нужно сделать так, чтобы ушедший
    # "за экран" игрок выходил с другой стороны


def check_move():
    if canvas.coords(player) == canvas.coords(exit):
        label.config(text="Победа!")


def key_pressed(event):
    if event.keysym == 'Up':
        move_wrap(player, (0, -step))
    # Здесь нужно дописать то, что нужно,
    # чтобы все остальные клавиши работали
    check_move()


master = tkinter.Tk()

step = 60
N_X = 10
N_Y = 10
canvas = tkinter.Canvas(master, bg='blue',
                        width=step * N_X, height=step * N_Y)

player_pos = (random.randint(0, N_X - 1) * step,
              random.randint(0, N_Y - 1) * step)
exit_pos = (random.randint(0, N_X - 1) * step,
            random.randint(0, N_Y - 1) * step)

player = canvas.create_oval((player_pos[0], player_pos[1]),
                            (player_pos[0] + step, player_pos[1] + step),
                            fill='green')
exit = canvas.create_oval((exit_pos[0], exit_pos[1]),
                          (exit_pos[0] + step, exit_pos[1] + step),
                          fill='yellow')
```

```
label = tkinter.Label(master, text="Найди выход")
label.pack()
canvas.pack()
master.bind("<KeyPress>", key_pressed)
master.mainloop()
```

Пока есть одна проблема: когда игрок находит выход, ничего не происходит. Он может гулять и дальше. Чтобы исправить это, можно связать с событием какую-нибудь другую функцию. Пусть эта функция ничего не делает. Определим ее и добавим в функцию `check_move` одну строчку:

```
def do_nothing(x):
    pass


def check_move():
    if canvas.coords(player) == canvas.coords(exit):
        label.config(text="Победа!")
        master.bind("<KeyPress>", do_nothing)
```

Обратите внимание, что функция `do_nothing` принимает на вход один аргумент. Функции, которые используются в команде `bind`, тоже всегда получают на вход один аргумент — описание события. Если бы мы определили функцию `do_nothing()` без параметров, то в процессе выполнения программы получили бы ошибку.

Теперь, попадая к выходу, игрок **теряет управление**, и нам остается только закрыть окно. Наверное, стоит добавить кнопку, которая позволит ему начать сначала.

Создание кнопки

Кнопка создается командой `tkinter.Button(...)`

В качестве параметров ей нужно передать **окно**, в котором будет создаваться кнопка; **текст**, который будет написан на кнопке и **функцию**, которая вызывается при ее нажатии.

Например так:

```
restart = tkinter.Button(master, text="Начать заново",
                         command=prepare_and_start)
restart.pack()
```

Теперь стоит перенести в отдельную функцию код, подготавливающий игровое поле:

```
def prepare_and_start():
    global player, exit
    player_pos = (random.randint(0, N_X - 1) * step,
                  random.randint(0, N_Y - 1) * step)
    exit_pos = (random.randint(0, N_X - 1) * step,
```

```
                random.randint(0, N_Y - 1) * step)
    player = canvas.create_oval(
        (player_pos[0], player_pos[1]),
        (player_pos[0] + step, player_pos[1] + step),
        fill='green')
    exit = canvas.create_oval(
        (exit_pos[0], exit_pos[1]),
        (exit_pos[0] + step, exit_pos[1] + step),
        fill='yellow')
    label.config(text="Найди выход!")
    master.bind("<KeyPress>", key_pressed)
```

Обратите внимание на строчку `global...`. Скоро мы узнаем, что это — глобальные переменные. Мы столкнемся с ними и поговорим о том, что использовать их надо *с умом*.

Однако в нашем случае без глобальных переменных трудно обойтись: информацию об игроке, выходе, холсте и т.д. пришлось бы передавать во все функции в качестве параметров. Это неудобно, а иногда и вовсе невозможно.

Например, команда `bind` работает с функциями, получающими ровно один аргумент — событие.

Поскольку наша программа невелика, мы решим эту проблему за счет глобальных переменных.

В реальных (больших) программах для этого используют классы, с которыми вы познакомитесь немного позже.

Основной код нашей программы теперь выглядит так:

```
step = 60  # Размер клетки
N_X = 10
N_Y = 10   # Размер сетки
master = tkinter.Tk()
label = tkinter.Label(master, text="Найди выход")
label.pack()
canvas = tkinter.Canvas(master, bg='blue',
                        height=N_X * step, width=N_Y * step)
canvas.pack()
restart = tkinter.Button(master, text="Начать заново",
                         command=prepare_and_start)
restart.pack()
prepare_and_start()
master.mainloop()
```

При попытке запустить новую программу вы сразу столкнетесь с новой проблемой: после нажатия **Начать заново** игрок и «выход» не исчезают. Нужно добавить в функцию `prepare_and_start` удаление всех старых объектов. К счастью, это можно сделать одной командой: `canvas.delete("all")`. Добавьте ее в вашу программу.

В нашу игру уже можно играть, но игроку чересчур легко живется: он даже проиграть не может! Давайте добавим препятствия: например, огонь, в который нельзя наступать.

Для этого придется переписать функцию `prepare_and_start`:

```
def prepare_and_start():
    global player, exit, fires
    canvas.delete("all")
    player_pos = (random.randint(0, N_X - 1) * step,
                  random.randint(0, N_Y - 1) * step)
    exit_pos = (random.randint(0, N_X - 1) * step,
                random.randint(0, N_Y - 1) * step)
    player = canvas.create_oval(
        (player_pos[0], player_pos[1]),
        (player_pos[0] + step, player_pos[1] + step),
        fill='green')
    exit = canvas.create_oval(
        (exit_pos[0], exit_pos[1]),
        (exit_pos[0] + step, exit_pos[1] + step),
        fill='yellow')
    N_FIRES = 6  # Число клеток, заполненных огнем
    fires = []
    for i in range(N_FIRES):
        fire_pos = (random.randint(0, N_X - 1) * step,
                    random.randint(0, N_Y - 1) * step)
        fire = canvas.create_oval(
            (fire_pos[0], fire_pos[1]),
            (fire_pos[0] + step, fire_pos[1] + step),
            fill='red')
        fires.append(fire)
    label.config(text="Найди выход!")
    master.bind("<KeyPress>", key_pressed)
```

И функцию, проверяющую результат хода:

```
def check_move():
    if canvas.coords(player) == canvas.coords(exit):
        label.config(text="Победа!")
        master.bind("<KeyPress>", do_nothing)
    for f in fires:
        if canvas.coords(player) == canvas.coords(f):
            label.config(text="Ты проиграл!")
            master.bind("<KeyPress>", do_nothing)
```

Игра уже почти как настоящая. Осталось два штриха:

1. Улучшить графику и

2. Добавить еще врагов

Добавление графики

На холст (Canvas) можно добавить любую картинку. В зависимости от типа изображения код будет немного варьироваться, мы будем рассматривать работу с изображениями в формате gif. Сначала картинку нужно загрузить с помощью функции `tkinter.PhotoImage`, а затем создать на холсте:

```
player_pic = tkinter.PhotoImage(file="doctor.gif")
player = canvas.create_image((player_pos[0], player_pos[1]),
                             image=player_pic, anchor='nw')
```

Параметр `anchor='nw'` означает, что в указанную первым параметром координату помещается левый верхний (буквально — северо-западный, по-английски — north-west) угол картинки. Если этот параметр не указать, то картинка будет центрирована по заданной координате. Можно добавить картинки для всех объектов в основную часть кода и в функцию `prepare_and_start()`.

В основную часть кода:

```
player_pic = tkinter.PhotoImage(file="images/doctor.gif")
exit_pic = tkinter.PhotoImage(file="images/tardis.gif")
fire_pic = tkinter.PhotoImage(file="images/fire.gif")
enemy_pic = tkinter.PhotoImage(file="images/dalek.gif")
```

И в функцию `prepare_and_start`:

```
def prepare_and_start():
    global player, exit, fires
    canvas.delete("all")
    player_pos = (random.randint(0, N_X - 1) * step,
                  random.randint(0, N_Y - 1) * step)
    player = canvas.create_image(
        (player_pos[0], player_pos[1]), image=player_pic, anchor='nw')
    exit_pos = (random.randint(0, N_X - 1) * step,
                random.randint(0, N_Y - 1) * step)
    exit = canvas.create_image(
        (exit_pos[0], exit_pos[1]), image=exit_pic, anchor='nw')
    N_FIRES = 6  # Число клеток, заполненных огнем
    fires = []
    for i in range(N_FIRES):
        fire_pos = (random.randint(0, N_X - 1) * step,
                    random.randint(0, N_Y - 1) * step)
        # fire = canvas.create_oval((fire_pos[0],fire_pos[1]),
        # (fire_pos[0] + step, fire_pos[1] + step), fill='red')
        fire = canvas.create_image(
            (fire_pos[0], fire_pos[1]), image=fire_pic, anchor='nw')
        fires.append(fire)
    label.config(text="Найди выход!")
    master.bind("<KeyPress>", key_pressed)
```

Картинки можно выбрать свои — они должны быть в формате gif и иметь размер step*step пикселей. Желательно так же делать их на прозрачном фоне.

Ну и последний штрих. Добавим настоящих врагов, которые тоже могут двигаться. Создадим их в функции `prepare_and_start()`, немного модифицировав последнюю:

```
def prepare_and_start():
    global player, exit, fires, enemies
    canvas.delete("all")
    player_pos = (random.randint(0, N_X - 1) * step,
                  random.randint(0, N_Y - 1) * step)
    player = canvas.create_image(player_pos, image=player_pic, anchor='nw')
    exit_pos = (random.randint(0, N_X - 1) * step,
                random.randint(0, N_Y - 1) * step)
    exit = canvas.create_image(exit_pos, image=exit_pic, anchor='nw')
    N_FIRES = 6 #Число клеток, заполненных огнем
    fires = []
    for i in range(N_FIRES):
        fire_pos = (random.randint(0, N_X - 1) * step,
                    random.randint(0, N_Y - 1) * step)
        fire = canvas.create_image(fire_pos, image=fire_pic, anchor='nw')
        fires.append(fire)
    N_ENEMIES = 4 #Число врагов
    enemies = []
    for i in range(N_ENEMIES):
        enemy_pos = (random.randint(0, N_X - 1) * step,
                     random.randint(0, N_Y - 1) * step)
        enemy = canvas.create_image(enemy_pos, image=enemy_pic, anchor='nw')
        enemies.append((enemy, random.choice([always_right, random_move])))
    label.config(text="Найди выход!")
    master.bind("<KeyPress>", key_pressed)
```

Каждый враг в нашей программе будет представлен парой (объект на Canvas + функция движения). Определим для начала две таких функции:

```
def always_right():
    return (step, 0)


def random_move():
    return random.choice([(step, 0), (-step, 0), (0, step), (0, -step)])
```

Модифицируем функцию `key_pressed`: ее нужно дополнить перемещением врагов — вот таким фрагментом кода:

```
for enemy in enemies:
    direction = enemy[1]() # вызвать функцию перемещения у "врага"
    move_wrap(enemy[0], direction) # произвести  перемещение
```

Кроме того, нужно переписать функцию `check_move`:

```
def check_move():
    if canvas.coords(player) == canvas.coords(exit):
        label.config(text="Победа!")
        master.bind("<KeyPress>", do_nothing)
    for f in fires:
        if canvas.coords(player) == canvas.coords(f):
            label.config(text="Ты проиграл!")
            master.bind("<KeyPress>", do_nothing)
    for e in enemies:
        if canvas.coords(player) == canvas.coords(e[0]):
            label.config(text="Ты проиграл!")
            master.bind("<KeyPress>", do_nothing)
```

Справка

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках сервиса, принадлежат АНО ДПО «Образовательные технологии Яндекса». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «Образовательные технологии Яндекса».
Пользовательское соглашение.

© 2018 – 2024 ООО «Яндекс»