Случайные числа в Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

Для генерации случайных чисел нужно подключить библиотеку random:

from random import \*

Для решения учебных задач достаточно знания двух функций:

x = random()

n = randint(a, b)

x примет случайное дробное значение от 0 включительно до 1 не включительно.

n примет случайное целое значение от a до b включительно, где a и b — целые числа.

Игра поймай шарик на Python

Информатика. Язык программирования Python в курсе информатики. Все классы.



from tkinter import \*
from random import \*

screen\_width = 600
screen\_height = 400
r\_min = 20
r\_max = 50
dt = 10  # микросекунд
#ball\_sprite\_filename = "ball\_sprite.png"
scores\_format = 'очки: %d'

class MainWindow:
    def \_\_init\_\_(self, root):
        global canvas
        canvas = Canvas(root)
        canvas["width"] = screen\_width
        canvas["height"] = screen\_height
        canvas.pack()

        self.ball = Ball.generate\_random\_ball()
        self.scores = 0
        self.scores\_text = canvas.create\_text(screen\_width - 50, 10,
                                              text=scores\_format%self.scores)

        self.game\_cycle()  # запуск игрового цикла
        canvas.bind("<Button-1>", self.mouse\_click)

    def mouse\_click(self, event):
        """ Проверяем, далеко ли шарик, и, если в него попали, то "лопаем" его,
            создаём новый шарик, а за старый начисляем очки.
        """
        if self.ball.check\_contact(event.x, event.y):
            self.scores += self.ball.scores
            canvas.itemconfig(self.scores\_text, text=scores\_format%self.scores)
            self.scores\_text
            self.ball.destroy()
            self.ball = Ball.generate\_random\_ball()

    def game\_cycle(self, \*ignore):
        canvas.after(dt, self.game\_cycle)  # перезапуск цикла
        if self.ball is not None:
            self.ball.move()

class Ball:
    def \_\_init\_\_(self, x, y, r, Vx=0, Vy=0):
        self.x, self.y, self.r = x, y, r
        self.Vx, self.Vy = Vx, Vy
        self.avatar = canvas.create\_oval(x-r, y-r, x+r, y+r, fill="red")
        self.scores = 10 + r\_max - r

    def check\_contact(self, x, y):
        l = ((self.x - x)\*\*2 + (self.y - y)\*\*2)\*\*0.5
        return l <= self.r

    def destroy(self):
        canvas.delete(self.avatar)

    def move(self):
        """ сдвинуть шарик на его скорость """
        # FIXME
        self.x += 1
        canvas.move(self.avatar, 1, 0)
        pass

    @classmethod
    def generate\_random\_ball(cls):
        r = randint(r\_min, r\_max)
        x = randint(r, screen\_width-r-1)
        y = randint(r, screen\_height-r-1)
        # FIXME: добавить генерацию случайной скорости
        return Ball(x, y, r)

root\_window = Tk()
#ball\_sprite = PhotoImage(file=ball\_sprite\_filename)
window = MainWindow(root\_window)
root\_window.mainloop()

Моделирование идеального газа на Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

from tkinter import \*
from random import \*

frame\_sleep\_time = 10 # задержка между кадрами в милисекундах
dt = 0.1 # квант игрового времени между кадрами
default\_radius = 20
atoms\_number = 20

class Atom:
    def \_\_init\_\_(self, canvas):
        self.r = default\_radius
        self.x, self.y = self.generate\_random\_ball\_coord()
        self.vx, self.vy = self.generate\_random\_ball\_velocity()
        self.avatar = canvas.create\_oval(self.x - self.r, self.y - self.r,
        self.x + self.r, self.y + self.r, fill='green')
        self.\_canvas = canvas

    def move(self):
        new\_x = self.x + self.vx\*dt
        new\_y = self.y + self.vy\*dt
        if new\_x < self.r or new\_x > screen\_width - self.r:
            new\_x = self.x # rolling back coordinate!
            self.vx = -self.vx
        if new\_y < self.r or new\_y > screen\_height - self.r:
            new\_y = self.y # rolling back coordinate!
            self.vy = -self.vy
        self.\_canvas.move(self.avatar, new\_x - self.x, new\_y - self.y)
        self.x, self.y = new\_x, new\_y

    def generate\_random\_ball\_coord(self):
        x = randint(self.r, screen\_width - self.r)
        y = randint(self.r, screen\_height - self.r)
        return x, y

    def generate\_random\_ball\_velocity(self):
        vx = randint(-10, +10)
        vy = randint(-10, +10)
        return vx, vy

def check\_collision(atom1, atom2):
    """определяет факт столкновения между атомами
    atom1, atom2 — экземпляры класса Atom"""
    return (atom1.x - atom2.x)\*\*2 + (atom1.y - atom2.y)\*\*2 <= (atom1.r + atom2.r)\*\*2

def collide(atom1, atom2):
    dx = atom2.x - atom1.x
    dy = atom2.y - atom1.y
    # n — единичный вектор от центра одного атома до центра другого,
    # он же вектор нормали плоскости столкновения
    nx = dx/(dx\*\*2 + dy\*\*2)\*\*0.5
    ny = dy/(dx\*\*2 + dy\*\*2)\*\*0.5

    # разложение скорости первого атома на компоненту продольную и поперечную оси столкновения
    v1\_parallel = atom1.vx\*nx + atom1.vy\*ny # скалярное произведение скорости atom1 на вектор n
    v1\_parallel\_x = v1\_parallel\*nx
    v1\_parallel\_y = v1\_parallel\*ny
    v1\_perpendicular\_x = atom1.vx - v1\_parallel\_x
    v1\_perpendicular\_y = atom1.vy - v1\_parallel\_y

    # разложение скорости второго атома на компоненту продольную и поперечную оси столкновения
    v2\_parallel = atom2.vx\*nx + atom2.vy\*ny # скалярное произведение скорости atom1 на вектор n
    v2\_parallel\_x = v2\_parallel\*nx
    v2\_parallel\_y = v2\_parallel\*ny
    v2\_perpendicular\_x = atom2.vx - v2\_parallel\_x
    v2\_perpendicular\_y = atom2.vy - v2\_parallel\_y

    # собираем обратно скорости атомов, сохраняя перпендикулярную компоненту скорости и обменивая поперечные
    atom1.vx = v1\_perpendicular\_x + v2\_parallel\_x
    atom1.vy = v1\_perpendicular\_y + v2\_parallel\_y
    atom2.vx = v2\_perpendicular\_x + v1\_parallel\_x
    atom2.vy = v2\_perpendicular\_y + v1\_parallel\_y

class GasStatistics:
    def \_\_init\_\_(self, canvas):
        self.\_canvas = canvas
        self.histogram\_bins = 10
        self.histogram\_v\_max = 10 # максимальная скорость отображаемая в гистограмме
        self.\_histogram = [0]\*self.histogram\_bins
        self.\_scale\_x = (int(canvas["height"]) - 40)/self.histogram\_v\_max
        self.\_scale\_y = 5
        self.\_x0 = 15 # местоположение оси координат для графика статистики
        self.\_y0 = int(canvas["height"]) - 20
        # рисуем на холсте оси координат
        canvas.create\_line(self.\_x0, self.\_y0, int(canvas["width"])-15, self.\_y0,
        width=2, fill="blue", arrow=LAST) # ось x
        canvas.create\_line(self.\_x0, self.\_y0, self.\_x0, 0 + 15, width=2,
        fill="blue", arrow=LAST) # ось y

    def screen\_xy(self, x, y):
        x = self.\_x0 + x\*self.\_scale\_x
        y = self.\_y0 - y\*self.\_scale\_y
        return x, y

    def calculate\_histogram(self, atoms):
        self.\_histogram[:] = [0]\*self.histogram\_bins
        histogram\_bin\_dv = self.histogram\_v\_max/self.histogram\_bins
        for atom in atoms:
            v = (atom.vx\*\*2 + atom.vy\*\*2)\*\*0.5
            # определение бина гистограммы, в который попадает данная скорость
            i = int(v/self.histogram\_v\_max\*self.histogram\_bins)
            i %= len(self.\_histogram) # чтобы не было вылета за пределы массива гистограммы
            self.\_histogram[i] += 1

    def show\_histogram(self):
        self.\_canvas.delete('bin')
        for bin in range(self.histogram\_bins):
            x, y = self.screen\_xy(bin, self.\_histogram[bin])
            self.\_canvas.create\_oval(x-5, y-5, x+5, y+5, fill='red', tag='bin')

def time\_event():
    # даём возможность подвинуться всем целям
    for atom in atoms:
        atom.move()
    # проверяем столкновения атомов друг с другом
    for i in range(len(atoms)):
        for k in range(i+1, len(atoms)):
            if check\_collision(atoms[i], atoms[k]):
                collide(atoms[i], atoms[k])
    stats.calculate\_histogram(atoms)
    stats.show\_histogram()
    gas\_canvas.after(frame\_sleep\_time, time\_event)

root = Tk()
gas\_canvas = Canvas(root, width=600, height=600)
gas\_canvas.pack(side='left')
screen\_width = int(gas\_canvas["width"])
screen\_height = int(gas\_canvas["height"])

statistics\_frame = Frame(root)
statistics\_canvas = Canvas(statistics\_frame, width=400, height=400)
freeze\_statistics\_button = Button(statistics\_frame, text="freeze")
statistics\_canvas.pack(side='top')
freeze\_statistics\_button.pack(side='top')
statistics\_frame.pack(side='left')

atoms = [Atom(gas\_canvas) for i in range(atoms\_number)]
stats = GasStatistics(statistics\_canvas)

time\_event() # начинаю циклически запускать таймер
root.mainloop()

Игра Tank Wars на Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

from tkinter import \*
from random import \*
from math import \*

frame\_sleep\_time = 5   # задержка между кадрами в милисекундах
dt = 0.01              # квант игрового времени между кадрами
g = 9.8                # гравитационная постоянная игры
screen\_width = 800     # ширина игрового экрана
screen\_height = 600    # высота игрового экрана

def create\_scores\_text():
    global scores\_text
    scores\_text = canvas.create\_text(60, 12, text="Scores: " + str(scores),
                                     font="Sans 18")

def change\_scores\_text():
    canvas.itemconfigure(scores\_text, text="Scores: " + str(scores))

def screen\_x(\_physical\_x):
    return round(\_physical\_x)

def screen\_y(\_physical\_y):
    return screen\_height - round(\_physical\_y)

def physical\_x(\_screen\_x):
    return \_screen\_x

def physical\_y(\_screen\_y):
    return screen\_height - \_screen\_y

class Shell:
    def \_\_init\_\_(self, x, y, vx, vy):
        self.r = 5  # отображаемый радиус при полёте
        self.x, self.y = x, y
        self.vx, self.vy = vx, vy
        self.avatar = canvas.create\_oval(screen\_x(self.x) - self.r, screen\_y(self.y) - self.r,
                                         screen\_x(self.x) + self.r, screen\_y(self.y) + self.r, fill="black")

    def move(self):
        new\_x = self.x + self.vx\*dt
        new\_y = self.y + self.vy\*dt - g\*dt\*\*2/2
        self.vy -= g\*dt
        if new\_x < self.r or new\_x > screen\_width - self.r:
            new\_x = self.x  # rolling back coordinate!
            self.vx = -self.vx
        if new\_y - self.r <= 0 or new\_y + self.r > screen\_height:
            new\_y = self.y  # rolling back coordinate!
            self.vy = -self.vy
        canvas.coords(self.avatar, screen\_x(new\_x) - self.r, screen\_y(new\_y) - self.r,
                      screen\_x(new\_x) + self.r, screen\_y(new\_y) + self.r)
        self.x, self.y = new\_x, new\_y

class Bullet(Shell):
    """Пуля - самый лёгкий из снарядов.
    Никогда не заканчивается, ничего не стоит, наносит минимальное поражение."""
    def \_\_init\_\_(self, x, y, vx, vy):
        super().\_\_init\_\_(x, y, vx, vy)
        self.radius\_of\_destruction = 1
        self.damage = 100
        canvas.coords(self.avatar, screen\_x(self.x) - self.r, screen\_y(self.y) - self.r,
                      screen\_x(self.x) + self.r, screen\_y(self.y) + self.r)

class Tank:
    def \_\_init\_\_(self):
        self.health = 100
        self.r = 20
        self.x = randint(self.r, screen\_width - self.r)
        self.y = ground[self.x]
        self.lx = 40
        self.ly = 40
        self.barrel\_avatar = canvas.create\_line(screen\_x(self.x), screen\_y(self.y), screen\_x(self.x + self.lx),
                                                screen\_y(self.y - self.ly), fill="black", width=3)
        self.body\_avatar = canvas.create\_oval(screen\_x(self.x - self.r), screen\_y(self.y - self.r),
                                              screen\_x(self.x + self.r), screen\_y(self.y + self.r),
                                              fill="grey")

    def shoot(self):
        vx = self.lx
        vy = self.ly
        return Bullet(self.x + self.lx, self.y + self.ly, vx, vy)

    def aim(self, x, y):
        l = ((x - self.x)\*\*2 + (y - self.y)\*\*2)\*\*0.5
        self.lx = 40\*(x - self.x)/l
        self.ly = 40\*(y - self.y)/l
        canvas.coords(self.barrel\_avatar, screen\_x(self.x), screen\_y(self.y),
                      screen\_x(self.x + self.lx), screen\_y(self.y + self.ly))

    def check\_collision(self, shell):
        """ Проверяет, попал ли снаряд в танк.
            x, y — координаты снаряда.
            возвращает True или False"""
        return (shell.x - self.x)\*\*2 + (shell.y - self.y)\*\*2 <= (self.r + shell.r)\*\*2

    def get\_damage(self, shell):
        self.health -= shell.damage

class Ground:
    def \_\_init\_\_(self, ground\_color="green"):
        self.ground\_color = ground\_color
        self.ground\_height = [200 + sin(x\*2\*pi/300)\*100 for x in range(screen\_width)]
        self.avatars = []
        for x in range(screen\_width):
            ground\_column = canvas.create\_rectangle(screen\_x(x), screen\_y(0), screen\_x(x),
                                                    screen\_y(self.ground\_height[x]),
                                                    fill=ground\_color, outline=ground\_color)
            self.avatars.append(ground\_column)

    def \_\_getitem\_\_(self, item):
        return self.ground\_height[item]

    def check\_collision(self, shell):
        """ Проверяет, попал ли снаряд в землю.
            x, y — координаты снаряда.
            возвращает True или False"""
        min\_x = max(round(shell.x) - shell.r, 0)  # предохранение от выхода за границу экрана при анализе столкновения
        max\_x = min(round(shell.x) + shell.r, screen\_width)
        for x in range(min\_x, max\_x):
            y = self.ground\_height[x]
            if (x - shell.x)\*\*2 + (y - shell.y)\*\*2 <= shell.r\*\*2:
                return True
        return False

    def boom(self, shell):
        shell.avatar = None

def time\_event():
    global scores, current\_bullet

    # если снаряд существует, то он летит
    if current\_bullet:
        current\_bullet.move()
        collision = ground.check\_collision(current\_bullet)  # проверка, не столкнулся ли снаряд с землёй
        for target in tanks:  # или с одним из танков
            if target.check\_collision(current\_bullet):
                collision = True

        if collision:
            print("BOOOM!")
            ground.boom(current\_bullet)
            for target in tanks:
                target.get\_damage(current\_bullet)
            current\_bullet = None

        # scores += 1   # FIXME: сделать много очков
        # change\_scores\_text()

    canvas.after(frame\_sleep\_time, time\_event)

def mouse\_move(event):
    # целимся пушкой на курсор
    current\_tank.aim(physical\_x(event.x), physical\_y(event.y))

def mouse\_click(event):
    global current\_bullet
    if current\_bullet:
        canvas.delete(current\_bullet.avatar)
    current\_bullet = current\_tank.shoot()

root = Tk()
canvas = Canvas(root, width=screen\_width, height=screen\_height)
canvas.pack()

scores = 0  # FIXME очки для всех

ground = Ground()
tanks = [Tank() for i in range(2)]
current\_tank = tanks[0]
current\_bullet = None

create\_scores\_text()
canvas.bind('<Button-1>', mouse\_click)
canvas.bind('<Motion>', mouse\_move)
time\_event()  # начинаю циклически запускать таймер
root.mainloop()

1. Игра Жизнь Джона Конвея на Python
2. Информатика. Язык программирования Python в курсе информатики. Все классы.
3. Клеточный автомат "Игра Жизнь" Джона Конвея на языке Python 3:
4. # License: GPLv3
\_\_author\_\_ = "Timofey Khirianov"
from tkinter import \*

frame\_sleep\_time = 100   # задержка между кадрами в милисекундах

cell\_width = 10
cell\_height = 10
cells\_horizontal\_number = 50
cells\_vertical\_number = 50
max\_physical\_x = cells\_horizontal\_number
max\_physical\_y = cells\_vertical\_number
screen\_width = cell\_width \* cells\_horizontal\_number    # ширина игрового экрана
screen\_height = cell\_height \* cells\_vertical\_number    # высота игрового экрана

def screen\_x(\_physical\_x):
    return round(\_physical\_x \* cell\_width)

def screen\_y(\_physical\_y):
    return screen\_height - round(\_physical\_y \* cell\_height)

def physical\_x(\_screen\_x):
    return \_screen\_x / cell\_width

def physical\_y(\_screen\_y):
    return (screen\_height - \_screen\_y) / cell\_height

def cell\_color(symbol):
    colors = {0: 'white', 1: 'green', ' ': None}
    return colors[symbol]

def cell\_outline\_color(symbol):
    colors = {0: 'lightgray', 1: 'lightgray', ' ': None}
    return colors[symbol]

class Field:
    def \_\_init\_\_(self, field\_file, canvas):
        """загружает поле с клетками из файла"""
        self.\_canvas = canvas
        with open(field\_file) as file:
            self.matrix = [None] \* cells\_vertical\_number
            self.avatars = [None] \* cells\_vertical\_number
            for yi in range(cells\_vertical\_number):
                self.matrix[yi] = [None] \* cells\_horizontal\_number
                self.avatars[yi] = [None] \* cells\_horizontal\_number
                line = file.readline().rstrip()
                line += ' '\*(cells\_horizontal\_number - len(line))
                for xi in range(cells\_horizontal\_number):
                    # любой символ, кроме пробела — значикт соотв. клетка жива
                    is\_cell\_alive = 0 if line[xi] == ' ' else 1
                    self.matrix[yi][xi] = is\_cell\_alive
                    self.avatars[yi][xi] = canvas.create\_rectangle(screen\_x(xi), screen\_y(yi),
                                                                   screen\_x(xi+1), screen\_y(yi+1),
                                                                   fill=cell\_color(is\_cell\_alive),
                                                                   outline=cell\_outline\_color(is\_cell\_alive))

    def calculate(self):
        """  """
        # рассчитываем матрицу состояний клеток на следующем шаге
        new\_matrix = [[0]\*cells\_horizontal\_number for i in range(cells\_vertical\_number)]
        for yi in range(1, cells\_vertical\_number-1):
            for xi in range(1, cells\_horizontal\_number-1):
                # подсчитаем количество живых соседей
                number\_of\_neighbours = 0
                for i in range(-1, 2):
                    for j in range(-1, 2):
                        number\_of\_neighbours += self.matrix[yi+i][xi+j]
                number\_of\_neighbours -= self.matrix[yi][xi]
                cell\_is\_alive = self.matrix[yi][xi]
                if (cell\_is\_alive and number\_of\_neighbours == 2) or number\_of\_neighbours == 3:
                    new\_matrix[yi][xi] = 1
                else:
                    new\_matrix[yi][xi] = 0
        # копируем рассчитанную матрицу в self.matrix
        for yi in range(1, cells\_vertical\_number-1):
            for xi in range(1, cells\_horizontal\_number-1):
                if self.matrix[yi][xi] != new\_matrix[yi][xi]:
                    self.matrix[yi][xi] = new\_matrix[yi][xi]
                    self.\_canvas.delete(self.avatars[yi][xi])
                    self.avatars[yi][xi] = self.\_canvas.create\_rectangle(screen\_x(xi), screen\_y(yi),
                                                                         screen\_x(xi+1), screen\_y(yi+1),
                                                                         fill=cell\_color(new\_matrix[yi][xi]),
                                                                         outline=cell\_outline\_color(new\_matrix[yi][xi]))

def time\_event():
    global scores
    # перевычислить состояние поля с клетками
    field.calculate()
    canvas.after(frame\_sleep\_time, time\_event)

def mouse\_move(event):
    pass

def mouse\_click(event):
    pass

if \_\_name\_\_ == "\_\_main\_\_":
    root = Tk()
    canvas = Canvas(root, width=screen\_width, height=screen\_height)
    canvas.pack()
    canvas.bind('<Motion>', mouse\_move)

    field = Field('map1.txt', canvas)

    time\_event()  # начинаю циклически запускать таймер
    root.mainloop()

Игра Arkanoid на Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

\_\_author\_\_ = 'Timofey Khirianov'

# GPLv3 license

from tkinter import \*
from random import \*
from math import \*

frame\_sleep\_time = 5   # задержка между кадрами в милисекундах
dt = 0.1              # квант игрового времени между кадрами

brick\_width = 40
brick\_height = 15
bricks\_horizontal\_number = 10
bricks\_vertical\_number = 20
max\_physical\_x = bricks\_horizontal\_number
max\_physical\_y = bricks\_vertical\_number
screen\_width = brick\_width\*bricks\_horizontal\_number    # ширина игрового экрана
screen\_height = brick\_height\*bricks\_vertical\_number    # высота игрового экрана

def screen\_x(\_physical\_x):
    return round(\_physical\_x\*brick\_width)

def screen\_y(\_physical\_y):
    return screen\_height - round(\_physical\_y\*brick\_height)

def physical\_x(\_screen\_x):
    return \_screen\_x/brick\_width

def physical\_y(\_screen\_y):
    return (screen\_height - \_screen\_y)/brick\_height

def create\_scores\_text():
    global scores\_text
    scores\_text = canvas.create\_text(60, 12, text="Scores: " + str(scores),
                                     font="Sans 18")

def change\_scores\_text():
    canvas.itemconfigure(scores\_text, text="Scores: " + str(scores))

def brick\_color(symbol):
    colors = {'r':'red', 'g':'green', 'b':'blue', 'y':'yellow', ' ':None}
    return colors[symbol]

class Bricks:
    def \_\_init\_\_(self, level\_file, canvas):
        """загружает карту из файла"""
        self.\_canvas = canvas
        with open(level\_file) as file:
            self.matrix = [None]\*bricks\_vertical\_number
            self.avatars = [None]\*bricks\_vertical\_number
            for yi in range(bricks\_vertical\_number):
                self.matrix[yi] = [None]\*bricks\_horizontal\_number
                self.avatars[yi] = [None]\*bricks\_horizontal\_number
                line = file.readline().rstrip()
                line = line + ' '\*(bricks\_horizontal\_number - len(line))
                for xi in range(bricks\_horizontal\_number):
                    color = brick\_color(line[xi])
                    if color != None:
                        self.matrix[yi][xi] = color
                        self.avatars[yi][xi] = canvas.create\_rectangle(screen\_x(xi), screen\_y(yi),
                                                                       screen\_x(xi+1), screen\_y(yi+1), fill=color)

    def check\_collision(self, ball):
        """ проверка, есть ли кирпич в левой, правой, верхней или нижней точке мячика """
        ball\_beat\_points = [(ball.x + ball.rx, ball.y), (ball.x + ball.rx, ball.y),
                            (ball.x, ball.y - ball.ry), (ball.x, ball.y - ball.ry)]
        for x, y in ball\_beat\_points:
            if self.matrix[floor(x)][floor(y)]:
                return True
        return False

    def delete\_brick(self, xi, yi):
        """ удалить кирпич """
        self.matrix[yi][xi] = None
        self.\_canvas.delete(self.avatars[yi][xi])
        self.avatars[yi][xi] = None

class Ball:
    def \_\_init\_\_(self, x, y, vx, vy):
        self.r = 5  # отображаемый радиус при полёте
        self.rx = self.r/brick\_width
        self.ry = self.r/brick\_height
        self.x, self.y = x, y
        self.old\_x, self.old\_y = x, y
        self.vx, self.vy = vx, vy
        self.avatar = canvas.create\_oval(screen\_x(self.x) - self.r, screen\_y(self.y) - self.r,
                                         screen\_x(self.x) + self.r, screen\_y(self.y) + self.r,
                                         fill="red")

    def move(self):
        new\_x = self.x + self.vx\*dt
        new\_y = self.y + self.vy\*dt
        if new\_x - self.rx <= 0:
            self.vx = abs(self.vx)
        elif new\_x + self.rx >= max\_physical\_x:
            self.vx = -abs(self.vx)
        if new\_y + self.ry >= max\_physical\_y:
            self.vy = -abs(self.vy)

        self.x, self.y, self.old\_x, self.old\_y = new\_x, new\_y, self.x, self.y

        canvas.coords(self.avatar, screen\_x(self.x) - self.r, screen\_y(self.y) - self.r,
                      screen\_x(self.x) + self.r, screen\_y(self.y) + self.r)
        global game\_over
        if new\_y <= 0:
            game\_over = True

class Racket:
    def \_\_init\_\_(self):
        self.lives = 3
        self.x = 0
        self.y = 0
        self.lx = 2  # ширина ракетки
        self.ly = 1  # высота ракетки
        self.avatar = canvas.create\_rectangle(screen\_x(self.x), screen\_y(self.y),
                                              screen\_x(self.x + self.lx),
                                              screen\_y(self.y + self.ly),
                                              fill="brown")

    def move(self, x, y):
        """ Двигает ракетку в указанную точку
            x, y — координаты мышки в экранных координатах
            """
        self.x = physical\_x(x)
        canvas.coords(self.avatar, screen\_x(self.x), screen\_y(self.y),
                      screen\_x(self.x + self.lx), screen\_y(self.y + self.ly))

    def check\_collision(self, ball):
        """ Проверяет, попал ли мяч в ракетку.
            ball — мячик.
            возвращает True или False"""
        return ball.y - ball.ry <= self.y + self.ly and self.x <= ball.x <= self.x + self.lx

def time\_event():
    global scores

    # если снаряд существует, то он летит
    if current\_ball:
        current\_ball.move()
        collision = racket.check\_collision(current\_ball)  # проверка, не столкнулся ли снаряд с землёй

        if collision:
            current\_ball.vy = abs(current\_ball.vy)
            scores += 1
            change\_scores\_text()
    if not game\_over:
        canvas.after(frame\_sleep\_time, time\_event)
    else:
        print("Game over!")

def mouse\_move(event):
    # целимся ракеткой на курсор мышки
    racket.move(event.x, event.y)

def mouse\_click(event):
    pass

if \_\_name\_\_ == "\_\_main\_\_":
    root = Tk()
    canvas = Canvas(root, width=screen\_width, height=screen\_height)
    canvas.pack()

    bricks = Bricks('map1.txt', canvas)
    scores = 0
    game\_over = False
    racket = Racket()
    current\_ball = Ball(racket.x+racket.lx/2, racket.y+racket.ly, 0.5, 1)

    create\_scores\_text()
    # canvas.bind('<Button-1>', mouse\_click)  # FIXME: раскомментировать, когда будет сделано залипание мяча на ракетке
    canvas.bind('<Motion>', mouse\_move)
    time\_event()  # начинаю циклически запускать таймер
    root.mainloop()

Игра Крестики нолики на Python

Информатика. Язык программирования Python в курсе информатики. Все классы.

Реализация игры "Крестики-нолики" с искусственным интеллектом (с рекурсивным анализом выигрышной стратегии):

# license: GPLv3
\_\_author\_\_ = "Timofey Khirianov"

def print\_field(field):
    for line in field[0:3], field[3:6], field[6:9]:
        print(\*line, sep = ' ')

def game\_over(field):
    """ определяет, может ли игра продолжаться.
        возвращает
            None, если игра ещё не закончена,
            'x', если выиграли крестики,
            'o', если выиграли нолики,
            '.', если ничья
    """
    a = field
    for i in range(3):  # определяю три одинаковых символа на горизонталях
        if a[3\*i+0] == a[3\*i+1] == a[3\*i+2] and a[3\*i+0] != '.':
            return a[3\*i+0]
    for j in range(3):  # определяю три одинаковых символа на вертикалях
        if a[0+j] == a[3+j] == a[6+j] and a[0+j] != '.':
            return a[0+j]
    # проверяем диагонали, нет ли на них трёх одинаковых символов
    if a[0] == a[4] == a[8] and a[0] != '.':
        return a[0]
    if a[2] == a[4] == a[6] and a[2] != '.':
        return a[2]
    # если управление пришло сюда, значит никто не выиграл
    # надо проверить, уже ничья или ещё продолжается игра
    for i in range(0, 9):
        if a[i] == '.':
            return None  # игра ещё не закончена
    return '.'  # ничья!

enemy\_symbol = {'x': 'o', 'o': 'x'}

def winning\_of\_position(field, player\_symbol):
    """ Определяет, выигрышная, проигрышная или нейтральная данная позиция
        +1 = выигрышная
        -1 = проигрышная
        0  = нейтральная
        field - список из 9 символов 'o', 'x' или '.'
    """
    game\_over\_state = game\_over(field)
    if game\_over\_state:
        if game\_over\_state == '.':  # ничья
            return 0
        elif game\_over\_state == player\_symbol:  # выиграл я!
            return +1
        else: # выиграл противник…
            return -1
    else:  # итак, игра ещё не закончилась — запускаем рекурсивный анализ ходов
        the\_worst\_position\_for\_enemy = None
        winning\_of\_the\_worst\_position\_for\_enemy = +2
        for pos in range(0, 9):
            if field[pos] == '.':
                field[pos] = player\_symbol  # делаю \_\_предполагаемый\_\_ ход
                winning\_of\_position\_after\_the\_turn = winning\_of\_position(field,
                                                                         enemy\_symbol[player\_symbol])
                if winning\_of\_position\_after\_the\_turn < winning\_of\_the\_worst\_position\_for\_enemy:
                    winning\_of\_the\_worst\_position\_for\_enemy = winning\_of\_position\_after\_the\_turn
                    the\_worst\_position\_for\_enemy = pos
                field[pos] = '.'  # Важно! Восстанавливаю пустоту позиции после анализа предполагаемого хода
        return -winning\_of\_the\_worst\_position\_for\_enemy  # -1, если худшая позиция для врага — выигрышная
                                                         # 0, если худшая позиция для врага — нейтральная
                                                         # +1, если худшая позиция для врага — проигрышная (ура!)

def ai\_choice(field, player\_symbol):
    """ Возвращает позицию, куда решил сходить Artificial Intelligence
        предполагается, что поле ещё не находится в состоянии game\_over,
        и ещё возможно совершить ход
    """
    the\_worst\_position\_for\_enemy = None
    winning\_of\_the\_worst\_position\_for\_enemy = +2
    for pos in range(0, 9):
        if field[pos] == '.':
            field[pos] = player\_symbol  # делаю \_\_предполагаемый\_\_ ход
            winning\_of\_position\_after\_the\_turn = winning\_of\_position(field,
                                                                     enemy\_symbol[player\_symbol])
            if winning\_of\_position\_after\_the\_turn < winning\_of\_the\_worst\_position\_for\_enemy:
                winning\_of\_the\_worst\_position\_for\_enemy = winning\_of\_position\_after\_the\_turn
                the\_worst\_position\_for\_enemy = pos
            field[pos] = '.'  # Важно! Восстанавливаю пустоту позиции после анализа предполагаемого хода
    assert the\_worst\_position\_for\_enemy != None, "Странно! Невозможно выбрать никакую позицию!.. %d"%the\_worst\_position\_for\_enemy
    return the\_worst\_position\_for\_enemy

def tournament():
    """ реализация игрового тура """
    field = ['.']\*9
    while True:
        pos = int(input("Введите номер позиции (число от 0 до 8)"))
        #pos = ai\_choice(field, 'x')
        field[pos] = 'x'
        print\_field(field)
        if game\_over(field):
            break
        pos = ai\_choice(field, 'o')
        print('Компьютер ходит в позицию:', pos)
        field[pos] = 'o'
        print\_field(field)
        if game\_over(field):
            break
    symbol = game\_over(field)
    if symbol == '.':
        print('Ничья!')
    else:
        print('Победили', symbol, '!')

if \_\_name\_\_ == "\_\_main\_\_":
    print("Привествуем Вас в игре крестики-нолики! С Вами играет Искусственный Интеллект!")
    tournament()