**Лабораторна робота №9**

**Тема. Робота з класами.**

**Мета роботи**. Освоїти створення 2D гри на Python 3.

**Зміст**.

1. Вивчення відомостей про бібліотеку PyGame.
2. Виконання роботи.
3. Отримання результату.

**Ключові положення.**

Є ігрове поле - простий прямокутник з твердими межами. Коли кулька торкається стінки або стелі, вона відскакує в іншу сторону. Якщо вона впаде на підлогу - ви програли. Щоб цього не сталося, внизу уздовж підлоги літає платформа, а ви керуєте нею за допомогою стрілок. Ваше завдання - підставляти платформу під кульку якомога довше. За кожний вдалий порятунок кульки ви отримуєте одне очко.

Щоб реалізувати таку логіку гри, потрібно передбачити такі сценарії поведінки:

1. гра починається;
2. кулька починає рухатися;
3. якщо натиснуті стрілки вліво або вправо - рухаємо платформу;
4. якщо кулька торкнулася стінок, стелі або платформи - робимо відскік;
5. якщо кулька торкнувся платформи - збільшуємо рахунок на одиницю;
6. якщо кулька впала на підлогу - виводимо повідомлення і закінчуємо гру.

Все це відбувається паралельно і незалежно один від одного. Тобто поки кулька літає, ми цілком можемо рухати платформу, а можемо і залишити її на місці. І коли кулька відскакує від стін, це теж не заважає іншим об'єктам рухатися і взаємодіяти між собою.

Виходить, що нам потрібно визначити три класи - платформу, саму кульку і рахунок, і визначити, як вони реагують на дії один одного. Поле нам самим визначати не потрібно - для цього є вже готова бібліотека. А потім в цих класах ми пропишемо методи - вони якраз і будуть відповідати за поведінку наших об'єктів.

Щоб у нас з'явилася графіка в грі, використовуємо бібліотеку Tkinter.

from tkinter import \* - Вона входить в набір стандартних бібліотек Python і дозволяє малювати найпростіші об'єкти - лінії, прямокутники, кола і фарбувати їх в різні кольори. Підключаємо модуль часу- import time для обмеження швидкості платформи, та модуль випадкових чисел-import random.

Щоб створити вікно, де буде видно графіка, використовують клас Tk (). tk = Tk() -він просто робить вікно, але без вмісту. Щоб з'явилося вміст, створюють полотно - видиму частину вікна. Робимо заголовок за допомогою властивості об’єкту tk.title('Game'). За допомогою властивості resizable не дозволяємо змінювати розміри вікна - tk.resizable(0, 0). Та розташуємо ігрове вікно зверху- tk.wm\_attributes('-topmost', 1).

Саме на ньому ми будемо малювати нашу гру. За полотно відповідає клас Canvas (), тому нам потрібно буде створити свій об'єкт з цього класу і далі вже працювати з цим об'єктом- canvas = Canvas(tk, width=500, height=400, highlightthickness=0). canvas.pack() – у кожного елемента гри, які ми бачимо будуть свої координати. tk.update()- команда оновлює вікно.

from tkinter import \*

import time

import random

tk = Tk()

tk.title('Game')

tk.resizable(0, 0)

tk.wm\_attributes('-topmost', 1)

canvas = Canvas(tk, width=500, height=400, highlightthickness=0)

canvas.pack()

tk.update()

Кулька повинна вміти:

1. задавати своє початкове положення і напрямок руху;
2. розуміти, коли він торкнувся платформи;
3. малювати сама себе і розуміти, коли потрібно відрендерити себе в новому положенні (наприклад, після відскоку від стіни).

Описуємо клас Ball, який відповідає за кульку.

class Ball:

def \_\_init\_\_(self, canvas, paddle, score, color):

self.canvas = canvas

self.paddle = paddle

self.score = score

self.id = canvas.create\_oval(10,10, 25, 25, fill=color)

Колір потрібен був для того, щоб ми зафарбувати усю кульку. Нова властивість id, в якої зберігається внутрішнє назва кульки а ще командою create\_oval створюється коло радіусом 15 пікселів і зафарбовується у потрібні кольори.

Початкова позиція кульки координата (245, 100), список для старту [-2, -1, 1, 2]. Та перемішуємо його, щоб кожний наступний раз кулька з’являлась у іншому місці. Задаємо перший із перемішаного вектор.

self.canvas.move(self.id, 245, 100)

starts = [-2, -1, 1, 2]

random.shuffle(starts)

self.x = starts[0]

self.y = -2

self.canvas\_height = self.canvas.winfo\_height()

self.canvas\_width = self.canvas.winfo\_width()

self.hit\_bottom = False

Початковий рух кульки завжди вниз тому зменшуємо його значення по осі у- self.y = -2. При цьому кулька визначає свої координати self.canvas\_height = self.canvas.winfo\_height() self.canvas\_width = self.canvas.winfo\_width(). self.hit\_bottom = False – властивість яка відповідає досягла чи ні дна кулька.

Створюємо функцію яка відповідає за торкання платформи.

def hit\_paddle(self, pos): -координати платформи будемо отримувати через об’єкт – paddle.

Створюємо умови, в яких перевіряємо чи співпадають координати платформи з координатами кульки if pos[2] >= paddle\_pos[0] and pos[0] <= paddle\_pos[2]: if pos[3] >= paddle\_pos[1] and pos[3] <= paddle\_pos[3]:

Якщо співпадають - додаємо до рахунку за допомогою команди self.score.hit()

Інакше не зіткнулись.

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]:

self.score.hit()

return True

return False

Створюємо рух кульки за допомогою функції draw. За допомогою self.canvas.move(self.id, self.x, self.y)- пересуваємо кульку на задані координати . Та запам’ятовуємо нові координати кульки. pos = self.canvas.coords(self.id)

Задаємо умови руху кульки.

1.Якщо кулька падає зверху if pos[1] <= 0- задаємо падіння на наступному кроці 2- self.y = 2.

2. Якщо кулька торкнулась правим нижнім кутом дна if pos[3] >= self.canvas\_height- помічаємо це в окремій змінній, та виводимо кількість очок.

3. Якщо відбулось торкання платформи-if self.hit\_paddle(pos) == True. Кулька відправляється вверх - self.y = -2.

4. Якщо кулька торкнулася лівої стінки if pos[0] <= 0, то кулька повинна рухатися вправо self.x = 2.

5. Якщо кулька торкнулася правої стінки if pos[2] >= self.canvas\_width, то кулька повинна рухатися вліво self.x = -2.

.

def draw(self):

self.canvas.move(self.id, self.x, self.y)

pos = self.canvas.coords(self.id)

if pos[1] <= 0:

self.y = 2

if pos[3] >= self.canvas\_height:

self.hit\_bottom = True

canvas.create\_text(250, 120, text='Ви програли', font=('Courier', 30), fill='red')

if self.hit\_paddle(pos) == True:

self.y = -2

if pos[0] <= 0:

self.x = 2

if pos[2] >= self.canvas\_width:

self.x = -2

Платформа повинна:

1. рухатися вліво або вправо в залежності від натиснутої стрілки;
2. розуміти, коли гра почалася і можна рухатися.

Намалюємо платформу, для цього створимо клас Paddle, який відповідає за платформи. Клас canvas говорить о том що платформа буде намальована, розміром 100 на 10 пікселів у вибраному кольорі. Стартові позиції [40, 60, 90, 120, 150, 180, 200] за допомогою random.shuffle(start\_1) перемішуємо та обираємо першу з перемішаних позицій self.starting\_point\_x = start\_1[0]. Переміщуємо платформу у стартову позицію та завмираємо self.canvas.move(self.id, self.starting\_point\_x, 300), self.x = 0.

class Paddle:

def \_\_init\_\_(self, canvas, color):

self.canvas = canvas

self.id = canvas.create\_rectangle(0, 0, 100, 10, fill=color)

start\_1 = [40, 60, 90, 120, 150, 180, 200]

random.shuffle(start\_1)

self.starting\_point\_x = start\_1[0]

self.canvas.move(self.id, self.starting\_point\_x, 300)

self.x = 0

Платформа, як і кулька узнає свою ширину self.canvas\_width = self.canvas.winfo\_width()

Задаємо обробник натискань:

1. Якщо натиснута стрілка вправо - виконується метод turn\_right ().
2. Якщо натиснута стрілка вліво - виконується метод turn\_left().
3. Якщо гра не почалася - чекаємо self.started = False.
4. Як тільки гравець натисне клавішу ENTER, гравець починається.

self.canvas\_width = self.canvas.winfo\_width()

self.canvas.bind\_all('<KeyPress-Right>', self.turn\_right)

self.canvas.bind\_all('<KeyPress-Left>', self.turn\_left)

self.started = False

self.canvas.bind\_all('<KeyPress-Return>', self.start\_game)

Описуємо рух платформи функціями

Якщо рухаємося вправо – додаємо по 2 пікселя по осі х.

Якщо рухаємося вліво – додаємо по -2 пікселя по осі х.

def turn\_right(self, event):

self.x = 2

def turn\_left(self, event):

self.x = -2

def start\_game(self, event):

self.started = True

Метод, який відповідає за рух платформи. Ми рухаємо платформу на задану кількість пікселів self.canvas.move(self.id, self.x, 0). Отримуємо координати холста pos = self.canvas.coords(self.id). Перевіряємо межі. Якщо платформа стикнулася з лівою межею – стоп if pos[0] <= 0,self.x = 0. Якщо платформа стикнулася з правою межею – стоп elif pos[2] >= self.canvas\_width, self.x = 0.

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

Від рахунку нам потрібно щоб він правильно реагував на дотик платформи, збільшував число очок і виводив їх на екран:

Опишемо клас Score, який відображує підрахунки. На початку він дорівнюється нулю self.score = 0. Створюємо напис, який показує поточний рахунок, робимо його потрібного кольору і запам'ятовуємо внутрішнє ім'я цього напису self.id = canvas.create\_text(450, 10, text=self.score, font=('Courier', 15), fill=color. Створимо функцію торкання платформи def hit(self). Збільшення рахунку на одиницю та записуємо нове значення рахунку.

class Score:

def \_\_init\_\_(self, canvas, color):

self.score = 0

self.canvas = canvas

self.id = canvas.create\_text(450, 10, text=self.score, font=('Courier', 15), fill=color

def hit(self):

self.score += 1

self.canvas.itemconfig(self.id, text=self.score)

Зараз все є для того, щоб почати писати саму гру. Сенс гри в тому, щоб не впустити кульку. Поки цього не сталося - все рухається, але як тільки кулька впала - потрібно показати повідомлення про кінець гри і зупинити програму.

1. Зробимо об’єкт – зелений розрахунок score = Score(canvas, 'green').
2. Зробимо об’єкт – білу платформу paddle = Paddle(canvas, 'White').
3. Зробимо об’єкт - червону кулю ball = Ball(canvas, paddle, score, 'red').
4. Організуємо цикл доки куля не торкнулась дна ми можемо рухати платформу.

score = Score(canvas, 'green')

paddle = Paddle(canvas, 'White')

ball = Ball(canvas, paddle, score, 'red')

while not ball.hit\_bottom:

if paddle.started == True:

ball.draw()

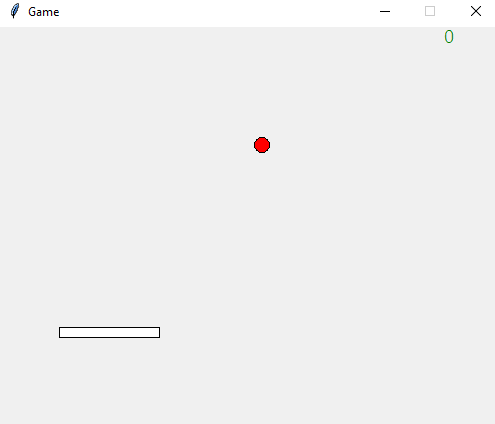
paddle.draw()

tk.update\_idletasks()

tk.update()

time.sleep(0.01)

time.sleep(3)



**Лабораторне завдання**

1. Розмалювати елементи гри у інші кольори.
2. Змінити розмір кулі та платформи.
3. Змінити швидкість руху платформи.
4. Змінити швидкість руху кульки.
5. Додати другу кульку.

Програма

