import turtle  
  
*# Create a screen*app = turtle.Screen()  
app.title("PingPong")  
app.bgcolor("blue")  
app.setup(width=900, height=600)  
app.tracer(0)  
  
*# Building the leftblock*leftblock = turtle.Turtle()  
leftblock.speed(0)  
leftblock.shape("square")  
leftblock.color("yellow")  
leftblock.shapesize(stretch\_wid=5, stretch\_len=1)  
leftblock.penup()  
leftblock.goto(-350, 0)  
  
*# Building the rightblock*rightblock = turtle.Turtle()  
rightblock.speed(0)  
rightblock.shape("square")  
rightblock.color("yellow")  
rightblock.shapesize(stretch\_wid=5, stretch\_len=1)  
rightblock.penup()  
rightblock.goto(350, 0)  
  
*# Building the ball*ball = turtle.Turtle()  
ball.speed(10)  
ball.shape("circle")  
ball.color("red")  
ball.penup()  
ball.goto(0, 0)  
ball\_x\_direct = 0.1  
ball\_y\_direct = 0.1  
  
score\_first = 0  
score\_second = 0  
  
*# Building 'write score' for update score*write\_score = turtle.Turtle()  
write\_score.color("Orange")  
write\_score.penup()  
write\_score.hideturtle()  
write\_score.goto(0, 260)  
write\_score.write(" Player A: {} Player B: {} ".format(score\_first, score\_second), align="center",  
 font=('Monaco', 24, "normal"))  
  
  
*# Move of the leftblock*def leftblocktop():  
 y = leftblock.ycor()  
 y = y + 50  
 leftblock.sety(y)  
  
  
def leftblockunder():  
 y = leftblock.ycor()  
 y = y - 50  
 leftblock.sety(y)  
  
  
*# Move of the rightblock*def rightblocktop():  
 y = rightblock.ycor()  
 y = y + 50  
 rightblock.sety(y)  
  
  
def rightblockunder():  
 y = rightblock.ycor()  
 y = y - 50  
 rightblock.sety(y)  
  
  
*# Control of blocks*app.listen()  
app.onkeypress(leftblocktop, 'w')  
app.onkeypress(leftblockunder, 's')  
app.onkeypress(rightblocktop, 'Up')  
app.onkeypress(rightblockunder, 'Down')  
  
while True:  
 app.update()  
  
 *# Move of the ball* ball.setx(ball.xcor() + ball\_x\_direct)  
 ball.sety(ball.ycor() + ball\_y\_direct)  
  
 *# Set up of the edges* if ball.ycor() > 290:  
 ball.sety(290)  
 ball\_y\_direct = -1 \* ball\_y\_direct  
 if ball.ycor() < -290:  
 ball.sety(-290)  
 ball\_y\_direct = ball\_y\_direct \* -1  
  
 if ball.xcor() > 390:  
 ball.goto(0, 0)  
 ball\_x\_direct = ball\_x\_direct \* -1  
 score\_first = score\_first + 1  
 write\_score.clear()  
 write\_score.write(" Player A: {} Player B: {} ".format(score\_first, score\_second),  
 align="center", font=('Monaco', 24, "normal"))  
  
 if ball.xcor() < -390:  
 ball.goto(0, 0)  
 ball\_x\_direct = ball\_x\_direct \* -1  
 score\_second = score\_second + 1  
 write\_score.clear()  
 write\_score.write(" Player A: {} Player B: {} ".format(score\_first, score\_second),  
 align="center", font=('Monaco', 24, "normal"))  
  
 q = leftblock.ycor()  
 o = rightblock.ycor()  
 if q > 290:  
 q = -250  
 leftblock.sety(q)  
 if q < -290:  
 q = 250  
 leftblock.sety(q)  
 if o > 290:  
 o = -250  
 rightblock.sety(o)  
 if o < -290:  
 o = 250  
 rightblock.sety(o)  
  
 *# Solving the collisions with touching blocks and ball* if (ball.xcor() > 340) and (ball.xcor() < 350) and (  
 ball.ycor() < rightblock.ycor() + 40 and ball.ycor() > rightblock.ycor() - 40):  
 ball.setx(340)  
 ball\_x\_direct = ball\_x\_direct \* -1  
  
 if (ball.xcor() < -340) and (ball.xcor() > -350) and (  
 ball.ycor() < leftblock.ycor() + 40 and ball.ycor() > leftblock.ycor() - 40):  
 ball.setx(-340)  
 ball\_x\_direct = ball\_x\_direct \* -1  
  
 *# Window after win Player A* if score\_first >= 5:  
 txt = turtle.Turtle()  
 txt.pen(shown=False)  
 txt.write('Player A win!', align='center', font=("Times New Roman", 40, 'bold'))  
 txt.penup()  
 txt.goto(-50, 50)  
 ball.hideturtle()  
  
 *# Window after win Player B* if score\_second >= 5:  
 txt = turtle.Turtle()  
 txt.pen(shown=False)  
 txt.write('Player B win!', align='center', font=("Times New Roman", 40, 'bold'))  
 txt.penup()  
 txt.goto(-50, 50)  
 ball.goto(0, 0)  
 ball.hideturtle()



from tkinter import \*

from tkinter import ttk

root = Tk()

root.geometry("400x450")

root.resizable(width = True, height = True)

root.title("Лабораторная работа6")

mainmenu = Menu(root)

root.configure(menu = mainmenu)

filemenu = Menu(mainmenu)

filemenu.add\_command(label = "Открыть файл")

filemenu.add\_command(label = "Сохранить файл")

mainmenu.add\_cascade(label = "File", menu = filemenu)

mainmenu.add\_cascade(label = "Settings")

label1 = Label(root, text = 'Как дела?', font = 'Calibri 20')

label1.grid(column= 0, row = 0)

label1.pack()

def click():

    print('Все хорошо')

btn = Button(root,  text = 'Все хорошо',

                    command = click,

                    font = 'TimesNewRoman 14',

                    bg = 'blue',

                    activebackground = 'green',

                    activeforeground = 'white')

btn.pack()

label2 = Label(root, text = 'Из какой ты группы?', font = 'Calibri 20')

label2.pack()

def dele():

    name.delete(0, END)

name = Entry(root, width= 20)

name.pack()

btn1 = Button(root, font = "TimesNewRoman 10", text = "Очистить поле ввода", command = dele)

btn1.pack()

enabled\_on = "Программа работает"

enabled\_off = "Программа не работает"

enabled = StringVar(value=enabled\_on)

enabled\_checkbutton = ttk.Checkbutton(textvariable=enabled, variable=enabled, offvalue=enabled\_off, onvalue=enabled\_on)

enabled\_checkbutton.pack(padx=6, pady=6, anchor=NW)

root.mainloop()

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

from tkinter import \*

from tkinter.ttk import Combobox

import tkinter as tk

#Функция для кнопки Джава

def clicked():

    lbl.configure(text="Джава или Ява? :)...")

#Функция для кнопки Пайтон

def clicked1():

    lbl.configure(text="Питон или Пайтон? :)...")

window = Tk()

window.title("Application PythonRu")

window.geometry('1280x450')

lbl = Label(window, text = "Какой язык программирования вы предпочитаете?", font=("Arial Bold", 20))

lbl.grid(column=0, row=0)

#Добавление кнопки Java

btn = Button(window, text="Java",font=("Arial Bold", 40),bg = 'white', fg = 'red', command=clicked)

btn.grid(column=0,row=1)

#Добавление кнопки Python

btn1 = Button(window, text="Python", font=("Arial Bold", 40), bg = 'white', fg = 'blue',command=clicked1)

btn1.grid(column=0,row=3)

#Добавление раскрывающейся строки

combo = Combobox(window, font=("Arial Bold", 15))

combo['values'] = ("User1","User2","User3")

combo.current(0)

combo.grid(column=3,row=0)

menu\_1 = tk.Menu(window)

window.config(menu=menu\_1)

# Создание разделов меню

file\_menu = tk.Menu(menu\_1)

edit\_menu = tk.Menu(menu\_1)

menu\_1.add\_cascade(label="File", menu=file\_menu)

menu\_1.add\_cascade(label="Edit", menu=edit\_menu)

# Добавление элементов в разделы меню

file\_menu.add\_command(label="New")

file\_menu.add\_command(label="Open")

file\_menu.add\_command(label="Save")

file\_menu.add\_separator()

file\_menu.add\_command(label="Exit", command=window.quit)

edit\_menu.add\_command(label="Cut")

edit\_menu.add\_command(label="Copy")

edit\_menu.add\_command(label="Paste")

window.mainloop()

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

from tkinter import \*

from random import \*

def finish():

root.destroy() # ручное закрытие окна и всего приложения

print("Закрытие приложения")

def howtoknow\_r(): # функция при нажатии кнопки Камень

global wins

rps = ['Rock', 'Paper', 'Scissors'] # Массив с возможными значениями

value = choice(rps) # с помощью функции рандом.choice выбираем случайное значение

labelText.configure(text=value) # передаем в labelText случайно выбранное значение

if value == 'Rock': # проверям по правилам камень-ножницы-бумага победили ли мы, выводи сообщение с результатом в labelwin

labelWin.configure(text='Draw...')

elif value == 'Paper':

labelWin.configure(text='You lost :(')

else:

labelWin.configure(text='You won!')

wins+=1

labelScore2.configure(text=wins)

def howtoknow\_p(): # все то же самое для бумаги

global wins

rps = ['Rock', 'Paper', 'Scissors']

value = choice(rps)

labelText.configure(text=value)

if value == 'Rock':

labelWin.configure(text='You won!')

wins+=1

labelScore2.configure(text=wins)

elif value == 'Paper':

labelWin.configure(text='Draw...')

else:

labelWin.configure(text='You lost :(')

def howtoknow\_s(): # все то же самое для ножниц

global wins

rps = ['Rock', 'Paper', 'Scissors']

value = choice(rps)

labelText.configure(text=value)

if value == 'Rock':

labelWin.configure(text='You lost :(')

elif value == 'Paper':

labelWin.configure(text='You won!')

wins+=1

labelScore2.configure(text=wins)

else:

labelWin.configure(text='Draw...')

global wins

wins = 0

root = Tk()

root.title('Rock Paper Scissors') # название окна

root.geometry('600x400') # размер окна

root.resizable(width=False, height=False) # чтобы окно невозможно было растягивать

root['bg'] = 'black'

labelScore1 = Label(root, text='Score: ', fg='white', font=('Times New Roman', 20), bg='black')

labelScore1.place(x=400, y=0)

labelScore2 = Label(root, text='', fg='white', font=('Times New Roman', 20), bg='black')

labelScore2.place(x=500, y=0)

labelComputer = Label(root, text='Computer chose: ', fg='white', font=('Times New Roman', 20), bg='black') # Показывает сообщение "что вывел компьютер", не меняется

labelComputer.place(x=50, y=150) # оно зафиксировано

labelText = Label(root, text='', fg='white', font=('Times New Roman', 20), bg='black') # наше рандомно значение

labelText.place(x=260, y=150)

labelWin = Label(root, text='', fg='white', font=('Times New Roman', 18), bg='black') # сообщение о победе/проигрыше

labelWin.place(x=260, y=200)

rock = Button(root, text='Rock', font=('Times New Roman', 20), bg='white', command=howtoknow\_r) # кнопка для выбора камня

rock.place(x=50, y=300)

scissors = Button(root, text='Scissors', font=('Times New Roman', 20), bg='white', command=howtoknow\_s) # кнопка для выбора ножниц

scissors.place(x=220, y=300)

paper = Button(root, text='Paper', font=('Times New Roman', 20), bg='white', command=howtoknow\_p) #кнопка для выбора бумаги

paper.place(x=420, y=300)

root.mainloop()

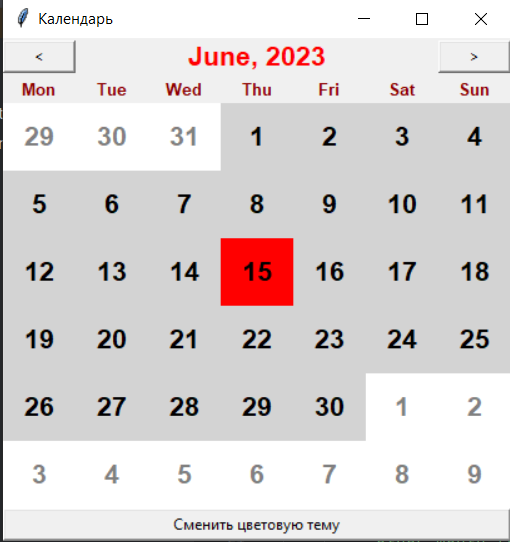
Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Автоматически созданное описание

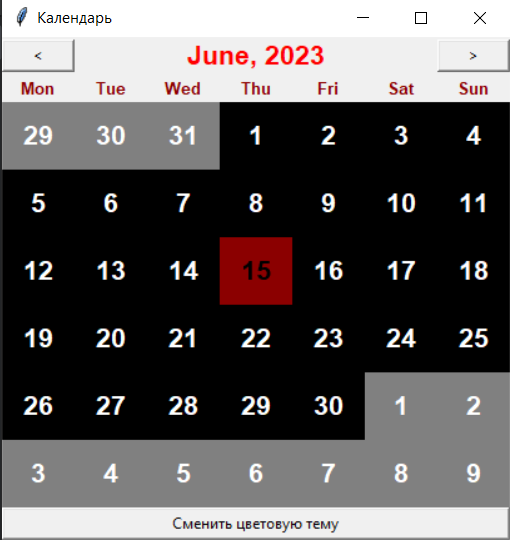
from tkinter import \*  
import calendar # для работы с календарем  
import datetime # для получения текущей даты и времени  
  
# Набор цветов для цветовой темы календаря  
color\_themes = {  
 'default': {  
 'bg': 'lightgray',  
 'fg': 'black',  
 'today\_bg': 'red',  
 'today\_fg': 'black',  
 'other\_month\_bg': 'white',  
 'other\_month\_fg': 'gray',  
 },  
 'dark': {  
 'bg': 'black',  
 'fg': 'white',  
 'today\_bg': 'darkred',  
 'today\_fg': 'black',  
 'other\_month\_bg': 'gray',  
 'other\_month\_fg': 'white',  
 }  
}  
  
current\_color\_theme = 'default' # Текущая цветовая тема  
  
# Функция для смены цветовой темы  
def change\_color\_theme():  
 global current\_color\_theme  
 if current\_color\_theme == 'default':  
 current\_color\_theme = 'dark'  
 else:  
 current\_color\_theme = 'default'  
 fill()  
  
  
# Функция back уменьшает значение переменной month на 1 и,  
# если month становится равным 0, устанавливает его в 12 и уменьшает значение переменной year на 1.  
def back():  
 global month, year  
 month -= 1  
 if month == 0:  
 month = 12  
 year -= 1  
 fill() # После обновления месяца и года вызывается функция `fill()` для обновления отображения календаря.  
  
  
# Функция next увеличивает значение переменной  
# month на 1 и, если month становится равным 13,  
# устанавливает его в 1 и увеличивает значение переменной year на 1.  
def next():  
 global month, year  
 month += 1  
 if month == 13:  
 month = 1  
 year += 1  
 fill()  
  
  
def fill(): # создаем функцию fill для заполнения ячеек с числами дней недель и заголовком  
 info\_label['text'] = calendar.month\_name[month] + ', ' + str(year) # заполняем текстом метку о месяце и годе  
 month\_days = calendar.monthrange(year, month)[1] # сохраняем в переменную month\_days информацию о кол-ве дней тек. месяца (получаем второе значение метода monthrange)  
 if month == 1:  
 back\_month\_days = calendar.monthrange(year - 1, 12)[1]  
 else: # сохраняем в переменную back\_month\_days информацию о кол-ве дней предыдущего месяца  
 back\_month\_days = calendar.monthrange(year, month - 1)[1]  
  
 week\_day = calendar.monthrange(year, month)[0] # сохраняем в week\_day ифнормацию о номере первого дня недели тек. месяца  
  
 for n in range(month\_days): # цикл для заполнения ячеек числами текущего месяца и окрашивания их  
 days[n + week\_day]['text'] = n + 1  
 days[n + week\_day]['fg'] = color\_themes[current\_color\_theme]['fg']  
 if year == now.year and month == now.month and n ==now.day: # если год, месяц и день равны текущим, то окрашиваем текущий день, также окрашиваем остальные ячейки  
 days[n + week\_day - 1]['bg'] = color\_themes[current\_color\_theme]['today\_bg']  
 days[n + week\_day - 1]['fg'] = color\_themes[current\_color\_theme]['today\_fg']  
 days[n + week\_day]['bg'] = color\_themes[current\_color\_theme]['bg']  
 else:  
 days[n + week\_day]['bg'] = color\_themes[current\_color\_theme]['bg']  
  
 for n in range(week\_day): # цикл для заполнения числами предыдущего месяца (вместе с текущим месяцем)  
 days[week\_day - n - 1]['text'] = back\_month\_days - n  
 days[week\_day - n - 1]['fg'] = color\_themes[current\_color\_theme]['other\_month\_fg']  
 days[week\_day - n - 1]['bg'] = color\_themes[current\_color\_theme]['other\_month\_bg']  
 for n in range(6 \* 7 - month\_days - week\_day): # цикл для заполнения числами след. месяца  
 days[week\_day + month\_days + n]['text'] = n + 1  
 days[week\_day + month\_days + n]['fg'] = color\_themes[current\_color\_theme]['other\_month\_fg']  
 days[week\_day + month\_days + n]['bg'] = color\_themes[current\_color\_theme]['other\_month\_bg']  
  
  
root = Tk()  
root.title('Календарь')  
days = [] # создание пустого списка days  
now = datetime.datetime.now() # сохраняем в переменную now информацию и текущей дате, времени  
year = now.year # текущий год  
month = now.month # текущий месяц  
  
back\_button = Button(text='<', command=back) # создаем кнопку для перехода на предыдущий месяц  
back\_button.grid(row=0, column=0, sticky=EW) # располагаем кнопку в первой ячейке условной сетки  
  
next\_button = Button(text='>', command=next) # создаем кнопку для перехода на след. месяц  
next\_button.grid(row=0, column=6, sticky=EW) # располагаем кнопку в последней ячейке первой строки условной сетки  
  
info\_label = Label(font='Arial 16 bold', fg='red') # создаем текстовую метку для информации о месяце и годе  
info\_label.grid(row=0, column=1, columnspan=5) # располагаем метку в 5 ячейках между кнопками перехода на пред. след. месяцы  
  
color\_button = Button(text='Сменить цветовую тему', command=change\_color\_theme) # создаем кнопку для изменения цветовой темы  
color\_button.grid(row=8, column=0, columnspan=7, sticky=EW) # располагаем кнопку в последней строке условной сетки  
  
for n in range(7): # создаем цикл для вывода сокращенных названий дней недели во второй строке  
 lbl = Label(text=calendar.day\_abbr[n], font='Arial 10 bold', fg='darkred')  
 lbl.grid(row=1, column=n)  
  
for row in range(6): # создаем цикл для заполнения условной сетки метками для чисел дней недель начиная с 3-й строки  
 for col in range(7):  
 lbl = Label(width=4, height=2, font='Arial 16 bold')  
 lbl.grid(row=row+2, column=col)  
 days.append(lbl)  
  
fill() # вызываем функцию fill, которая при запуске программы выведет текущий месяц и год  
  
root.mainloop()

Работа программы:

Обычная тема:



Темная тема:



# Великоднева Евгения, НПИбд-01-21, 14/06/23, 23:00

# Данная программа является главным меню для открытия двух мини-игр(крестики-нолики и рисование)

# Импортируем tkinter и классы с мини-играми

from tkinter import \*

from tttclss import Tictactoe

from drawingclass import Drawing

# Создаём окно приложения

root = Tk()

# Функции для вызова игр

def ttoe():

    root.geometry("600x600+450+100")

    delete\_main\_menu()

    kn = Tictactoe(root=root, menu=main\_menu)

    kn.tictactoe()

def draww():

    delete\_main\_menu()

    kn = Drawing(root, main\_menu)

    kn.draw()

# Создаём кнопки для выбора игр

lbl = Label(text="Choose game")

top\_frame = Frame()

ttt = Button(top\_frame, text="Tic-Tac-Toe", command=ttoe)

draw = Button(top\_frame, text="Drawing", command=draww)

draw.pack(side=LEFT)

ttt.pack(side=LEFT)

# Функция для создания главного меню(при переходе из меню игр)

def main\_menu():

    root.geometry("500x500+450+100")

    global top\_frame, lbl

    lbl.pack()

    top\_frame.pack()

# Функция для удаления главного меню(при переходе в меню игр)

def delete\_main\_menu():

    global lbl, top\_frame

    lbl.pack\_forget()

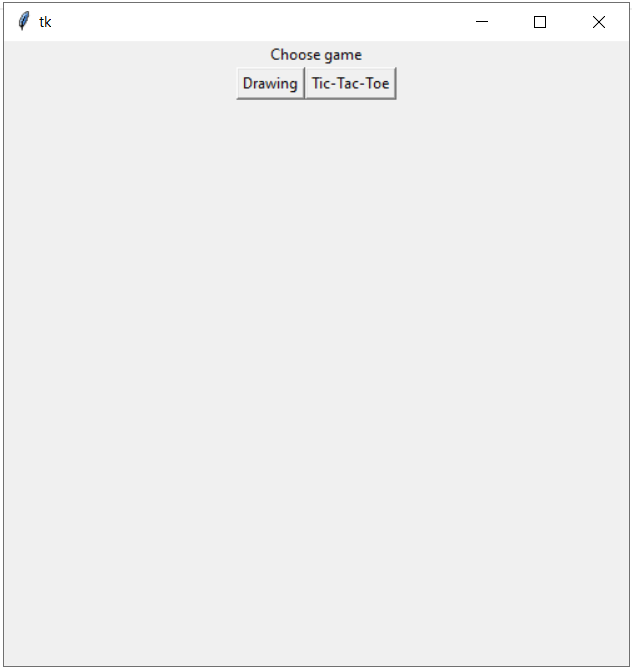
    top\_frame.pack\_forget()

# Функция для создания изначального меню

main\_menu()

# Запуск главного цикла обработки событий

root.mainloop()



1. Главное меню для обеих игр

# Код из файла с классом первой игры

from tkinter import \*

import random

# Класс для игры крестики-нолики

class Tictactoe:

    # Конструктор класса

    def \_\_init\_\_(self, root, menu):

        self.root = root

        # Переменная, которая показывает, какие клетки заняты

        self.numsarr = [0,1,2,3,4,5,6,7,8]

        # Переменные, которые подсчитывают количество крестиков и ноликов в каждом ряду, колонке и диагонали

        self.rows = [0,0,0]

        self.cols = [0,0,0]

        self.diag = [0,0]

        # Переменная для подсчёта ходов

        self.f = 0

        # Функция перехода в главное меню

        self.menu = menu

        self.strt\_rstrt = True

        self.restart\_btn = 0

    # Функция игры(Вызывается из главного меню)

    def tictactoe(self):

        # Переменная для понимания, какая функция должна быть у кнопки старт-рестарт

        self.strt\_rstrt = True

        # Функция для изменения функции кнопки старт(рестарт)

        def start\_restart(sr):

            if sr:

                start()

            else:

                restart()

        # Функция для перезапуска игры

        def restart():

            c.after(1000, delete\_canvas)

            restart\_vars()

            self.root.after(1100, start\_menu)

        # Функция для перехода в главное меню

        def main\_menu():

            c.after(600, delete\_canvas)

            players\_frame.pack\_forget()

            top\_frame.pack\_forget()

            self.menu()

        # Создаём фрейм для кнопок "старт-рестарт" и "главное меню"

        top\_frame = Frame(self.root)

        top\_frame.pack(side=TOP)

        # Функция для создания мини-меню(вышеуказанные кнопки)

        def make\_menu():

            self.restart\_btn = Button(top\_frame, text="Start", command = lambda: start\_restart(self.strt\_rstrt))

            self.restart\_btn.pack(side=LEFT)

            main\_menu\_btn = Button(top\_frame, text="Main menu", command=main\_menu)

            main\_menu\_btn.pack(side=LEFT)

        make\_menu()

        # Проверяем положение клика и определяем, в какой квадрат поставить крестик или нолик

        def check\_a\_b\_ab(y, x):

            if(x<=150):

                b = 0

            elif(x<=300):

                b = 1

            else:

                b = 2

            if(y<=150):

                a = 0

                ab = a + b

            elif(y<=300):

                a = 1

                ab = a + b + 2

            else:

                a = 2

                ab = a + b + 4

            return([a, b, ab])

        # Функция для "обнуления" переменных игры

        def restart\_vars():

            self.numsarr = [0,1,2,3,4,5,6,7,8]

            self.rows = [0,0,0]

            self.cols = [0,0,0]

            self.diag = [0,0]

            self.f = 0

        # Переменная и радиокнопки для выбора режима игры(1 или 2 игрока)

        choose\_mode = BooleanVar()

        players\_frame = Frame(self.root)

        players\_frame.pack()

        r1 = Radiobutton(players\_frame, text="One player", variable=choose\_mode, value=1, indicatoron = 0, background="pink")

        r2 = Radiobutton(players\_frame, text="Two players", variable=choose\_mode, value=0, indicatoron = 0, background="pink")

        # Холст для игры

        c = Canvas(self.root, width=450, height=450, bg="pink")

        # Функция для появления стартового меню

        def start\_menu():

            choose\_mode.set(1)

            self.restart\_btn.config(text="Start")

            self.strt\_rstrt = True

        start\_menu()

        # Функция для "уничтожения" стартового меню

        def destroy\_menu():

            r1.pack\_forget()

            r2.pack\_forget()

            self.restart\_btn.config(text="Restart")

            self.strt\_rstrt = False

        # Функция для создания полос на холсте и добавления холста в окно приложения

        def make\_canvas():

            c.pack(anchor='center', pady=25)

            c.create\_line(150, 0, 150, 450, fill="DeepPink4")

            c.create\_line(300, 0, 300, 450, fill="DeepPink4")

            c.create\_line(0, 150, 450, 150, fill="DeepPink4")

            c.create\_line(0, 300, 450, 300, fill="DeepPink4")

        # Функция для вывода сообщения о победе/ничьей

        def fin(n):

            if(n == 0):

                txt = "Draw"

            elif(n == 1):

                txt = "Player 1 wins!"

            else:

                txt = "Player 2 wins!"

            lblfr = Frame(c)

            lbl = Label(lblfr, text=txt)

            lbl.pack()

            c.create\_window(200, 200, window=lblfr)

        # Функция для очищения холста и удаления холста из окна

        def delete\_canvas():

            c.delete("all")

            c.pack\_forget()

        # Функция для создания сердечка(крестика/нолика)

        def make\_heart(a, b, color):

            heart = [75, 150, 150, 50, 110, 0, 75, 50, 40, 0, 0, 50]

            for i in range(0,len(heart)):

                if i%2 != 0:

                    heart[i]+=a\*150

                else:

                    heart[i]+=b\*150

            c.create\_polygon(heart, fill=color)

        # Функция для прикрепления функции к холсту

        def bind\_c():

            c.bind("<Button>", handle\_event)

        # Функция для проверки, победил ли кто-либо из игроков

        def check\_win(x, y, plus\_or\_minus):

                if plus\_or\_minus:

                    z = 1

                else:

                    z = -1

                self.rows[y] += z

                self.cols[x] += z

                if x+y==2:

                    self.diag[0] += z

                if x==y:

                    self.diag[1] += z

                if 3 in self.rows or 3 in self.cols or 3 in self.diag:

                    return True

                elif -3 in self.rows or -3 in self.cols or -3 in self.diag:

                    return True

                else:

                    return False

        # Функция для выбора случайной клетки для "нолика"(в игре на одного игрока)

        def choose\_random\_sq():

            n = random.randint(0,8-self.f)

            n = self.numsarr.pop(n)

            self.f += 1

            if(n<=2):

                a = 0

                b = n

            elif(n<=5):

                a = 1

                b = n - 3

            else:

                a = 2

                b = n - 6

            make\_heart(a, b, "blue")

            if check\_win(b, a, self.f%2==0):

                fin(2)

            else:

                c.after(500, bind\_c)

        # Функция для отслеживания координат мыши по клику установки крестиков/ноликов(Для режима одного игрока)

        def handle\_event(event):

            a, b, ab = check\_a\_b\_ab(event.y, event.x)

            if ab in self.numsarr:

                make\_heart(a, b, "DeepPink4")

                self.numsarr.remove(ab)

                c.unbind("<Button>")

                self.f += 1

                if check\_win(b, a, self.f%2==0):

                    fin(1)

                elif self.f == 9:

                    fin(0)

                else:

                    self.root.after(500, choose\_random\_sq)

        # Функция для отслеживания координат мыши по клику установки крестиков/ноликов(Для режима двух игроков)

        def two\_players(event):

            a, b, ab = check\_a\_b\_ab(event.y, event.x)

            if self.f%2 == 0:

                if ab in self.numsarr:

                    make\_heart(a, b, "DeepPink4")

                    self.numsarr.remove(ab)

                    self.f += 1

                    if check\_win(b, a, self.f%2 == 0) and self.f!=9:

                        fin(1)

                    elif self.f == 9:

                        fin(0)

            else:

                if ab in self.numsarr:

                    make\_heart(a, b, "blue")

                    self.numsarr.remove(ab)

                    self.f += 1

                    if check\_win(b, a, self.f%2 == 0):

                        fin(2)

        # Функция для начала игры

        def start():

            destroy\_menu()

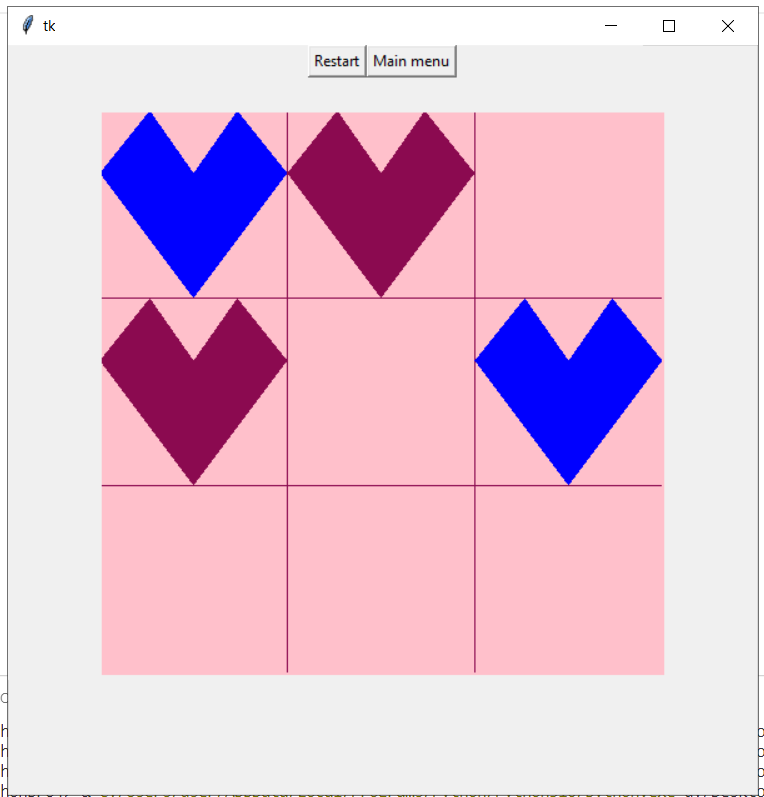
            make\_canvas()

            if choose\_mode.get():

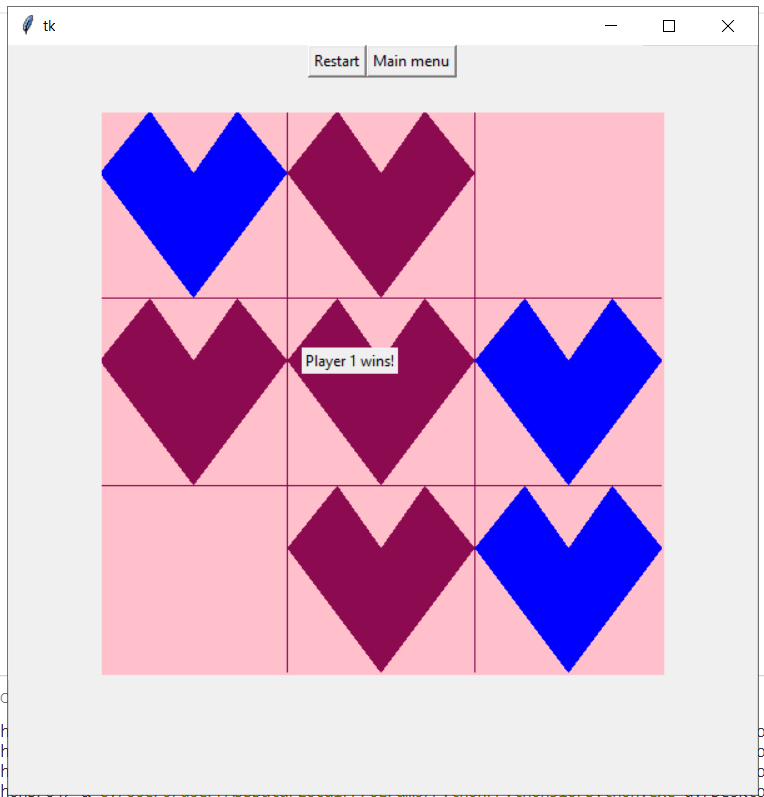
                c.bind("<Button>", handle\_event)

            else:

                c.bind("<Button>", two\_players)



2. Игра в крестики-нолики



3. Победа первого игрока

# Код из файла с классом второй игры

# Импортируем tkinter

from tkinter import \*

# Создаём класс для игры

class Drawing:

    # Конструктор класса

    def \_\_init\_\_(self, root, main\_menu):

        self.root = root

        self.main\_menu = main\_menu

    # Функция игры(именно её мы вызываем из главного меню)

    def draw(self):

        # Создаём необходимые переменные

        # Для изменения ширины кисти

        width = IntVar()

        width.set(10)

        # Для изменения цвета кисти

        color = StringVar()

        color.set("black")

        # Для выбора режима рисования

        draw\_erase = BooleanVar()

        draw\_erase.set(1)

        # Создаём фреймы для "левого меню", выбора режима выбора ширины(для удобного позиционирования)

        left\_menu = Frame(self.root)

        left\_menu = Frame(self.root)

        draw\_erase\_frame = Frame(left\_menu)

        draw\_or\_erase = Label(draw\_erase\_frame, text="Draw or erase")

        width\_frame = Frame(left\_menu)

        width\_choose = Label(width\_frame, text="Choose width")

        r1 = Radiobutton(draw\_erase\_frame, text="Draw", variable=draw\_erase, value=1, indicatoron = 0, background="DeepPink3", selectcolor="DeepPink2")

        r2 = Radiobutton(draw\_erase\_frame, text="Erase", variable=draw\_erase, value=0, indicatoron = 0, background="DeepPink3", selectcolor="DeepPink2")

        def erase\_all():

            c.delete("all")

        erase\_all\_btn = Button(left\_menu, text="Erase all", command=erase\_all)

        colors\_frame = Frame(left\_menu)

        colors\_frame\_right = Frame(colors\_frame)

        colors\_frame\_left = Frame(colors\_frame)

        # Создаём холст для рисования

        c = Canvas(self.root, width=600, height=600, background="pink")

        colors\_lbl = Label(colors\_frame, text="Choose color")

        # Создаём радиокнопки для выбора цвета

        red\_btn = Radiobutton(colors\_frame\_left, text="   Red  ", variable=color, value= "red", background="red", indicatoron=0, selectcolor="dark red")

        orange\_btn = Radiobutton(colors\_frame\_right, text="Orange", variable=color, value="orange", background="orange", indicatoron=0, selectcolor="dark orange")

        yellow\_btn = Radiobutton(colors\_frame\_left, text="Yellow", variable=color, value="yellow", background="yellow", indicatoron=0, selectcolor="light yellow")

        green\_btn = Radiobutton(colors\_frame\_right, text=" Green ", variable=color, value="green", background="green", indicatoron=0, selectcolor="dark green")

        blue\_btn = Radiobutton(colors\_frame\_left, text="  Blue  ", variable=color, value="blue", background="blue", indicatoron=0, selectcolor="dark blue")

        purple\_btn = Radiobutton(colors\_frame\_right, text=" Purple", variable=color, value="purple", background="purple", indicatoron=0, selectcolor="medium purple")

        black\_btn = Radiobutton(colors\_frame\_left, text=" Black ", variable=color, value="black", background="black", foreground="white", indicatoron=0, selectcolor="grey")

        white\_btn = Radiobutton(colors\_frame\_right, text=" White ", variable=color, value="white", background="white", indicatoron=0, selectcolor="grey")

        # Радиокнопки для выбора ширины кисти

        xs\_width = Radiobutton(width\_frame, text="Extra small", variable=width, value=5, indicatoron=0, selectcolor="pink")

        s\_width = Radiobutton(width\_frame, text="Small", variable=width, value=10, indicatoron=0, selectcolor="pink")

        m\_width = Radiobutton(width\_frame, text="Medium", variable=width, value=15, indicatoron=0, selectcolor="pink")

        l\_width = Radiobutton(width\_frame, text="Large", variable=width, value=20, indicatoron=0, selectcolor="pink")

        xl\_width = Radiobutton(width\_frame, text="Extra large", variable=width, value=30, indicatoron=0, selectcolor="pink")

        # Функция для добавления всех элементов на экран и установки размеров окна

        def start():

            self.root.geometry("900x700+200+50")

            left\_menu.pack(side=LEFT)

            draw\_erase\_frame.pack(pady=15)

            width\_frame.pack(pady=15)

            width\_choose.pack(pady=10)

            xs\_width.pack(side=LEFT)

            s\_width.pack(side=LEFT)

            m\_width.pack(side=LEFT)

            l\_width.pack(side=LEFT)

            xl\_width.pack(side=LEFT)

            colors\_frame.pack(side=TOP)

            colors\_lbl.pack(pady=10)

            colors\_frame\_right.pack(side=RIGHT)

            colors\_frame\_left.pack(side=LEFT)

            red\_btn.pack(side=TOP)

            orange\_btn.pack(side=TOP)

            yellow\_btn.pack(side=TOP)

            green\_btn.pack(side=TOP)

            blue\_btn.pack(side=TOP)

            purple\_btn.pack(side=TOP)

            black\_btn.pack(side=TOP)

            white\_btn.pack(side=TOP)

            c.pack(side=RIGHT)

            draw\_or\_erase.pack(pady=10)

            r1.pack(side=LEFT)

            r2.pack(side=RIGHT)

            erase\_all\_btn.pack(side=TOP, pady=15)

        # Функция для удаления всех параметров с экрана(при выходе из мини-игры) и вызова функции создания главного меню

        def finish():

            left\_menu.pack\_forget()

            c.pack\_forget()

            self.main\_menu()

        # Создание кнопки для перехода в главное меню

        main\_menu\_btn = Button(left\_menu, text="Main menu", command = finish)

        main\_menu\_btn.pack()

        start()

        # Функция для "рисования". Она фиксирует движения мыши и добавляет круги по всем координатам движения

        def handle\_event(event):

            if draw\_erase.get():

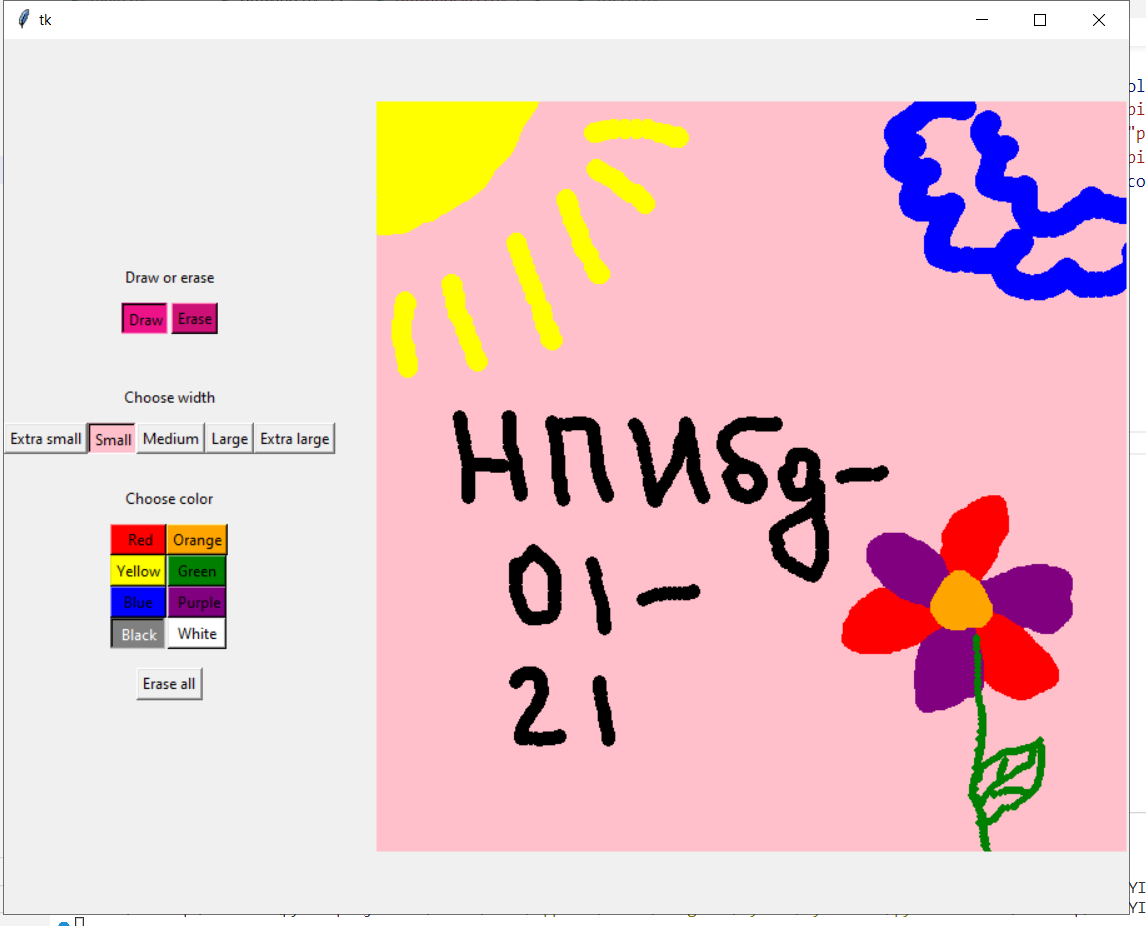
                c.create\_oval(event.x, event.y, event.x+width.get(), event.y+width.get(), fill=color.get(), outline="")

            else:

                c.create\_oval(event.x, event.y, event.x+width.get(), event.y+width.get(), fill="pink", outline="")

        # Добавление функции в холст

        c.bind("<B1-Motion>", handle\_event)



4. Рисование в мини-игре

from tkinter import \*

import tkinter.font as tkFont

import tkinter as tk

class QuizApp(Tk):

    questionNum = 0

    # Constructor

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.title("Dream Quizlet") # setting title of window app

        self.geometry("1080x720") # setting window size

        self['background']='#1c1d2b' # seting window background color

        self.resizable(False, False) # defining that window cannot be resized

        self.create\_widgets() # ask for creating main components(widgets) of the window app

    # This function gets questions/answers from the file "quizletQuestions.txt"

    def get\_quizlet\_questionsAnswers(self):

        quizletFile = open("quizletQuestions.txt", "r") # opening file and creating object to work with it

        lines = quizletFile.readlines() # reading lines(consisting of strings) and saving them in 'lines'

        self.num\_lines = len(lines) # calculating number of lines(questions and answers)

        self.questions = [] # array of questions

        self.answers = [] # array of answers

        for i in range(self.num\_lines):

            self.questions.append([])

            self.answers.append([])

        # filling arrays of questions and answers with

        for i in range(self.num\_lines):

            line = lines[i].strip()

            values = line.split()

            question = " ".join(values[:-1])

            answer = values[-1]

            self.questions[i].append(question)

            self.answers[i].append(answer)

            self.questions[i] = " ".join(self.questions[i])

            self.answers[i] = " ".join(self.answers[i])

    # This function creates the main components(widgets) of window app

    def create\_widgets(self):

        # Creating widgets

        # Creating main window sign

        myFont = tkFont.Font(family="Bauhaus 93", size = 46, slant = "italic", weight = "bold") # font using by this label's text

        self.welcomeSign = Label(text = "Dream\n\tQuizlet", highlightthickness = 0, font = myFont, fg = "#d2d952",

                       bg = "#1c1d2b", width = 15, height = 2)

        self.welcomeSign.place(x = 0, y = 0)

        # Creating question sign label

        menuLabelsFont1 = tkFont.Font(family="Britannic Bold", size = 16, slant = "italic", weight = "bold") # font using by this label's text

        self.questionSign\_label = tk.Label(self.master, text="Question {}".format(self.questionNum + 1), font = menuLabelsFont1, fg = "#d1a506",

                                      bg = "#1c1d2b", width = 18, height = 3, highlightthickness=0)

        self.questionSign\_label.place(x = 700, y = 50) # locating it

        # Ask for getting questions/answers from the file

        self.get\_quizlet\_questionsAnswers()

        # Creating question label

        self.question\_label = tk.Label(self.master, text=self.questions[0], font = menuLabelsFont1, fg = "#1c1d2b",

                                      bg = "#e6d8a5", width = 36, height = 7, borderwidth=2, relief="solid",

                                  highlightthickness=0, justify='center')

        self.question\_label.place(x = 600, y = 120) # locating it on the window

        entryLabelFont = tkFont.Font(family="Times New Roman", size = 16) # font using by this label's text

        # Creating answer checking entry

        self.answer\_entry = tk.Entry(self.master, font = entryLabelFont, width = 40)

        self.answer\_entry.place(x = 590, y = 300) # locating it

        # Creating answer checking button

        buttonFont = tkFont.Font(family = "Helvetica", size = 18, weight = "bold") #

        self.check\_button = tk.Button(self.master, text="Check", command=self.check\_answer)

        self.check\_button.config(font = buttonFont, width = 9,

                                  height = 0, fg = "#d2d952",bg = "#1c1d2b", borderwidth=2, relief="solid",

                                  highlightthickness=0)

        self.check\_button.place(x = 590, y = 350) # locating it

        self.canvas = Canvas(width=560, height=550, highlightthickness=0, bg = "#1c1d2b") # creating canvas-label for the mascot-images

        self.canvas.place(x = 5, y = 180) # locating on the window

        self.qbotStates = ["mascotPRESENT.png","mascotCORRECT.png","mascotWRONG.png"] # creating array of names of the image-files(which will be used by 'canvas')

        self.myImage = PhotoImage(file = self.qbotStates[0]) # Creating mascot-image on the application window

        self.canvas.create\_image(5, 25, anchor = NW, image = self.myImage) # defining the image used by 'canvas'

    # This function is used for checking the answer which is received from 'answer\_entry'-bar

    def check\_answer(self):

        # Checking if it's the last question

        if (self.questionNum + 1) == self.num\_lines:

            # if it's true then:

            self.questionSign\_label.config(text = "") # cleaning the text of 'questionSign\_label'

            self.myImage.config(file = self.qbotStates[0]) # setting default mascot-image

            self.question\_label.config(text = "Thanks for game!") # outputing the "Thanks"-answer in 'question\_label'

        else:

            # if it's false then:

            answer = self.answer\_entry.get() # catch the answer from the 'answer\_entry'-bar

            # if input-anwer is correct (congruent with answers[questionNum]) then:

            if answer == self.answers[self.questionNum]:

                self.myImage.config(file = self.qbotStates[1]) # setting correct-answer image

                self.answer\_entry.delete(0, 'end') # clearing the 'answer\_entry'-bar

                self.questionNum += 1 # incrementing the number of question(to get following question)

                self.questionSign\_label.config(text="Question {}".format(self.questionNum + 1)) # changing the current number of question with questionNum

                self.question\_label.config(text=self.questions[self.questionNum]) # changing current question to the following one

            else:

                self.answer\_entry.delete(0, 'end') # clearing the 'answer\_entry'-bar

                self.myImage.config(file = self.qbotStates[2]) # setting incorrect-answer image

                self.questionNum += 1 # changing the current number of question with questionNum

                self.questionSign\_label.config(text="Question {}".format(self.questionNum)) # changing the current number of question with questionNum

                self.question\_label.config(text=self.questions[self.questionNum]) # changing current question to the following one

app = QuizApp()

app.mainloop()

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание