ЛАБОРАТОРНАЯ РАБОТА

«РАЗРАБОТКА ИГРОВЫХ ПРОГРАММ В КОНСОЛЬНОМ ПРИЛОЖЕНИИ»

1. Игра «Крестики-нолики»

Описание программы: логическая игра между двумя противниками на квадратном поле 3 на 3 клетки. Один из игроков играет «крестиками», второй — «ноликами». Все действия игры происходят в терминале. Пользователю необходимо выбрать на поле незанятые ячейки, которые отмечены цифрами.

ЗАДАНИЕ: Отладьте код программы, зафиксируйте работу программы скриншотами в отчете. Далее модифицируйте программу так, чтобы:

а) для двух игроков поле игры было 5 на 5;

б) на поле 3 на 3 человек играл против компьютера.

Код программы:

```
print("*" * 10, " Игра Крестики-нолики для двух игроков ", "*" * 10)
board = list(range(1, 10))
def draw_board(board):
    print("-" * 13)
    for i in range(3):
        print("|", board[0 + i * 3], "|", board[1 + i * 3], "|", board[2 + i * 3], "|")
        print("-" * 13)
def take_input(player_token):
    valid = False
    while not valid:
        player_answer = input("Куда поставим " + player_token + "? ")
        try:
            player_answer = int(player_answer)
        except:
            print("Некорректный ввод. Вы уверены, что ввели число?")
            continue
        if 1 <= player_answer <= 9:
            if str(board[player_answer - 1]) not in "XO":
                board[player_answer - 1] = player_token
                valid = True
            else:
                print("Эта клетка уже занята!")
        else:
            print("Некорректный ввод. Введите число от 1 до 9.")
def check_win(board):
    win_coord = ((0, 1, 2), (3, 4, 5), (6, 7, 8), (0, 3, 6), (1, 4, 7), (2, 5, 8), (0,
4, 8), (2, 4, 6))
    for each in win_coord:
        if board[each[0]] == board[each[1]] == board[each[2]]:
            return board[each[0]]
    return False
def main(board):
    counter = 0
    win = False
    while not win:
        draw_board(board)
        if counter % 2 == 0:
            take_input("X")
        else:
            take_input("O")
        counter += 1
        if counter > 4:
            tmp = check_win(board)
            if tmp:
                print(tmp, "выиграл!")
                win = True
```

```
                break
            if counter == 9:
                print("Ничья!")
                break
        draw_board(board)
    main(board)
    input("Нажмите Enter для выхода!")
```

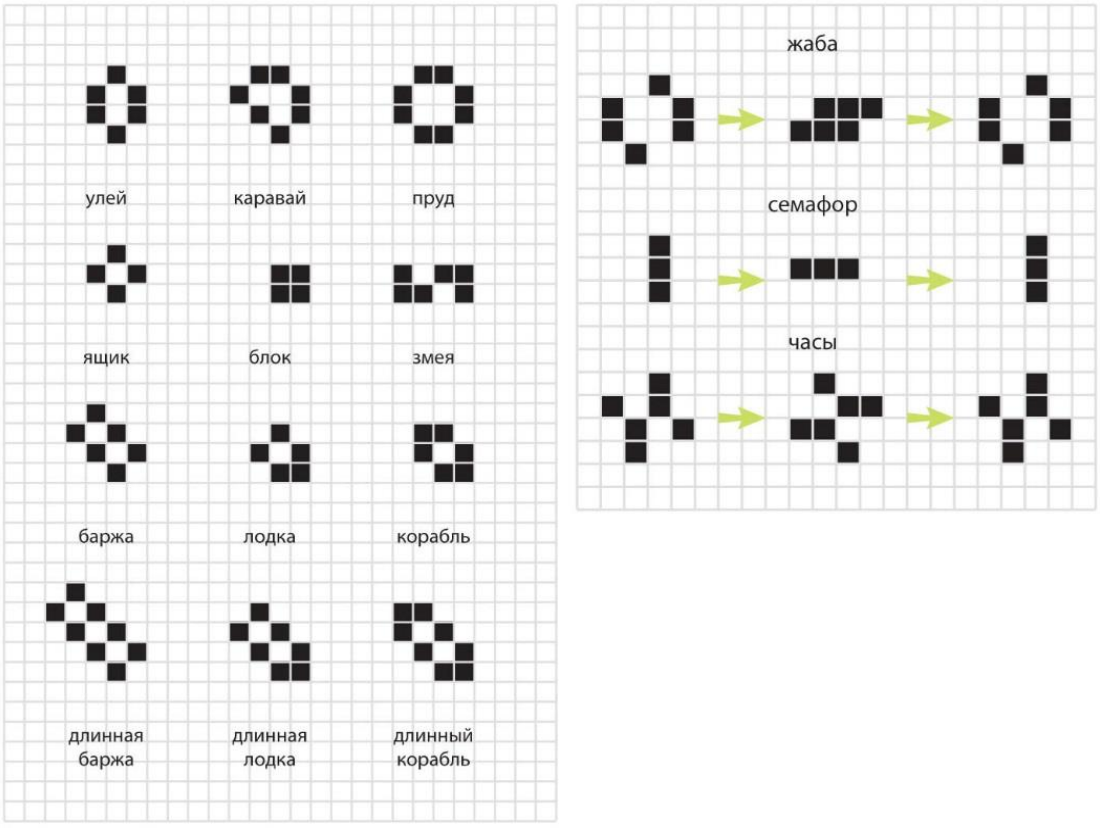
2. Игра «Жизнь»

Игра моделирует жизнь поколений гипотетической колонии живых клеток, которые выживают, размножаются или погибают. В соответствии со следующими правилами.

1. Клетка выживает, если она имеет двух или трех соседей из восьми возможных.
2. Клетка погибает от изоляции, если у нее нет соседей или только один сосед.
3. Клетка погибает от перенаселения, если у нее 4 и более соседей.
4. В любой пустой позиции, у которой ровно 3 соседа, в следующем поколении появляется новая клетка.

Игра «Жизнь» наглядно показывает возможности ***клеточных автоматов***. ***Клеточные автоматы*** — дискретная модель, состоящая из регулярной решетки ячеек (не только квадратных, но и, например, шестиугольных), каждая из которых может пребывать в одном из конечного числа состояний (например, 2 состояния – жив или мертв).

Правила «Жизни» были опубликованы кембриджским математиком Джоном Конвеем в 1970 году и сразу сделали клеточные автоматы невероятно популярными. В дальнейшем игра получила название «Жизнь Конвея».



Типичные структуры, возникающие в «Жизни» Конвея. Если игра начинается со случайной конфигурации, скорее всего, она закончится появлением устойчивого набора таких живучих форм. Но в общем случае эволюция системы непредсказуема, и чтобы выяснить, чем закончится такая «Жизнь», нужно ее «прожить», то есть смоделировать.

Для работы программы нам понадобятся системные библиотеки: time, sys, os, random.

Поскольку работа в консоли отличается в разных ОС, перед вызовом системных функций ОС по работе с консолью будем делать проверки, на какой ОС выполняется программа.

В самом начале импортируем библиотеки и опишем константы приложения:

```
import time
import os
import random
import sys

CHAR_LIFE="* " #Символ живой клетки
CHAR_EMPTY=". "#Символ пустой/мертвой клетки
CONST_GENERATION = 5000 # количество поколений в игре
CONST_TIME_SLEEP = 1 / 5.0 # время задержки вывода поля игры по умолчанию 1/5
секунды
ERR_CLS_OS_Unsupported="Невозможно очистить терминал. Ваша операционная система
не поддерживается.\n\r"
ERR_RESIZE_OS_Unsupported="Невозможно изменить размер терминала. Ваша
операционная система не поддерживается.\n\r"
ERR_NOT_Integer="Введено недопустимое целое значение."
ERR_Integer_not_in_range="Введенное число должно быть в диапазоне от {0} до {1}."
STR_INFO="Поколение {0} - для выхода из программы нажмите <Ctrl-C>\n\r"
STR_Input_rows="Введите количество строк в игре (10-60): "
STR_Input_cols="Введите количество столбцов в игре  (10-118): "
STR_PRESS_Enter_TO_EXIT="Нажмите <Enter> для завершения или r для новой игры: "
```

Далее напишем функцию очистки консоли.

```
def clear_console():
    """
    Очищает консоль с помощью системной команды в зависимости от операционной
системы пользователя.

    """
    if sys.platform.startswith('win'):
        os.system("cls")
    elif sys.platform.startswith('linux'):
        os.system("clear")
    elif sys.platform.startswith('darwin'):
        os.system("clear")
    else:
        print(ERR_CLS_OS_Unsupported)
```

Следующая функция изменяет размер терминала (консоли) – ширину и высоту:

```
def resize_console(rows, cols):
    """
    Изменяет размер консоли до размера rows x cols.

    :param rows: Int – количество строк для изменения размера консоли.
```

```
    :param cols: Int – количество столбцов для изменения размера консоли.
    """
    if cols < 32:
        cols = 32

    if sys.platform.startswith('win'):
        command = "mode con: cols={0} lines={1}".format(cols + cols, rows + 5)
        os.system(command)
    elif sys.platform.startswith('linux'):
        command = "\x1b[8;{rows};{cols}t".format(rows=rows + 3, cols=cols + cols)
        sys.stdout.write(command)
    elif sys.platform.startswith('darwin'):
        command = "\x1b[8;{rows};{cols}t".format(rows=rows + 3, cols=cols + cols)
        sys.stdout.write(command)
    else:
        print(ERR_RESIZE_OS_Unsupported)
```

Напишем функцию, которая генерирует случайную колонию на матрице консоли. В нашем примере, вероятность живой клетки на поле консоли 1/8.

```
def create_initial_grid(rows, cols):
    """
   Создает случайный список списков  (двумерный список/массив), содержащий 1 и 0, представляющий ячейки в игре «Жизнь» Конвея.

    :param rows: Int – количество строк в сетке Game of Life.
    :param cols: Int – количество столбцов в сетке Game of Life.
    :return: Int[][] – список списков, содержащий 1 для живых и 0 для мертвых ячеек.
    """
    grid = []
    for row in range(rows):
        grid_rows = []
        for col in range(cols):
            # Сгенерируем случайное число и на его основе решим, добавлять ли в сетку живую или мертвую ячейку.
            if random.randint(0, 7) == 0:
                grid_rows += [1]
            else:
                grid_rows += [0]
        grid += [grid_rows]
    return grid
```

Подготовим функцию вывода игрового поля на консоль:

```
def print_grid(rows, cols, grid, generation):
    """
    Распечатывает на консоли поле игры.

    :rows: Int – количество строк в поле.
    :cols: Int – количество столбцов в поле.
    :grid: Int[][] – Список списков, который используется для представления поля.
```

```
    :generation: Int — Номер текущего поколения поля.
    """

    clear_console()

    # Одна строка вывода используется для уменьшения мерцания, вызванного печатью
нескольких строк.
    output_str = ""

    #Соберем выходную строку и затем распечатайте ее на консоли.
    output_str += STR_INFO.format(generation)
    for row in range(rows):
        for col in range(cols):
            if grid[row][col] == 0:
                output_str += CHAR_EMPTY
            else:
                output_str += CHAR_LIFE
        output_str += "\n\r"
    print(output_str, end=" ")
```

Теперь напишем функцию, которая создает новое поколение, анализируя каждую клетку в текущем поколении и вычисляя какие клетки в следующем останутся, умрут или появятся новые.

```
def create_next_grid(rows, cols, grid, next_grid):
    """
    Анализирует текущее поколение на поле Игры и определяет, какие клетки живут и
умирают в следующем.

    :rows: Int — количество строк в поле игры
    :cols: Int — количество столбцов в поле игры
    :grid: Int[][] — Список списков, который будет использоваться для
представления  поля Игры текущего поколения.
    :next_grid: Int[][] — Список списков, который будет использоваться для
представления поля Игры следующего поколения.
    """

    for row in range(rows):
        for col in range(cols):
            # Получим количество живых ячеек, соседних с ячейкой grid[row][col]
            live_neighbors = get_live_neighbors(row, col, rows, cols, grid)

            # Если количество окружающих живых ячеек < 2 или > 3, мы делаем
мертвой ячейку  grid[row][col]
            if live_neighbors < 2 or live_neighbors > 3:
                next_grid[row][col] = 0
            # Если количество окружающих живых ячеек равно 3, а ячейка
grid[row][col] ранее была мертвой,
            # то делаем ее живой
            elif live_neighbors == 3 and grid[row][col] == 0:
                next_grid[row][col] = 1
```

```
            # Если количество окружающих живых ячеек равно 3, а ячейка
grid[row][col] жива, то сохраняем ее живой.
            else:
                next_grid[row][col] = grid[row][col]
```

Функция подсчета живых соседей клетки:

```
def get_live_neighbors(row, col, rows, cols, grid):
    """
    Подсчитывает количество живых ячеек, окружающих ячейку grid[row][cell].

    :param row: Int — строка клетки.
    :param col: Int — столбец клетки.
    :param rows: Int — количество строк в поле Игры.
    :param cols: Int — количество столбцов в поле Игры.
    :param grid: Int[][] — Список списков, который используется для представления
поля Игры.
    :return: Int — итговый результат: количество живых ячеек, окружающих ячейку
grid[row][cell]

    """

    life_sum = 0
    for i in range(-1, 2):
        for j in range(-1, 2):
            # Обязательно посчитайте центральную ячейку, расположенную в
grid[row][cell]
            if not (i == 0 and j == 0):
                #Используем оператор остатка от деления (%)
                life_sum += grid[((row + i) % rows)][((col + j) % cols)]
    return life_sum
```

Напишем функцию, которая проверяет, а были ли изменения в текущем и следующем новом поколении. Это поможет нам понять, когда можно завершать игру (если изменений нет).

```
def grid_changing(rows, cols, grid, next_grid):
    """
    Проверяет, совпадает ли текущее поле игры с полем следующего поколения.

    :param rows: Int — количество строк в игре.
    :param cols: Int — количество столбцов в  игре .
    :paramgrid: Int[][] — Список списков, который используется для представления
поля игры текущего поколения.
    :param next_grid: Int[][] — Список списков, который будет использоваться для
представления поля следующего поколения Игры.
    сетка
    :return: Логическое значение — совпадает ли сетка текущего поколения с сеткой
следующего поколения.
    """

    for row in range(rows):
```

```
        for col in range(cols):
            # Если ячейка grid[row][col] не равна next_grid[row][col]
            if not grid[row][col] == next_grid[row][col]:
                return True
    return False
```

Вспомогательная функция для ввода целого значения:

```
def get_integer_value(prompt, low, high):
    """
    Запрашивает у пользователя целочисленный ввод между заданными границами
нижнего и верхнего пределов.

    :param prompt: String – строка, запрашивающая у пользователя о вводе
    :param low: Int – нижняя граница, в пределах которой пользователь должен
оставаться.
    :param high: Int – верхняя граница, в пределах которой должен оставаться
пользователь.
    :return: Допустимое входное значение, введенное пользователем.
    """
    while True:
        try:
            value = int(input(prompt))
        except ValueError:
            print(ERR_NOT_Integer)
            continue
        if value < low or value > high:
            print(ERR_Integer_not_in_range.format(low, high))
        else:
            break
    return value
```

Функция для запуска игры:

```
def run_game():
    """
    Запрашивает у пользователя исходные данные для настройки игры «Жизнь» для
запуска заданного количества поколений.
    """
    clear_console()

    # Вводим количество строк и столбцов в поле игры.
    rows = get_integer_value(STR_Input_rows, 10, 60)
    clear_console()
    cols = get_integer_value(STR_Input_cols, 10, 118)
    resize_console(rows, cols)

    # Создаем начальное поле игры
    current_generation = create_initial_grid(rows, cols)
    next_generation = create_initial_grid(rows, cols)
```

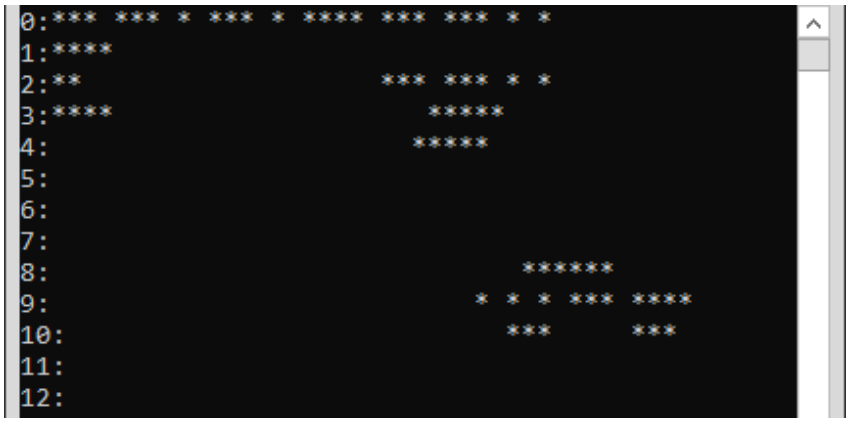
```
    # Основной цикл игры
    gen = 1
    for gen in range(1, CONST_GENERATION + 1):
        if not grid_changing(rows, cols, current_generation, next_generation):
            break
        print_grid(rows, cols, current_generation, gen)
        create_next_grid(rows, cols, current_generation, next_generation)
        time.sleep(CONST_TIME_SLEEP)
        current_generation, next_generation = next_generation, current_generation

    print_grid(rows, cols, current_generation, gen)
    return input(STR_PRESS_Enter_TO_EXIT)
```

И наконец, пишем код основной программы, которая в цикле запускает функцию run_game():

```
# Основная программа - запускает в цикле функцию run_game()
run = "r"
while run == "r":
    out = run_game()
    run = out
```

ЗАДАНИЕ. Отладьте код программы «Игра Жизнь», зафиксируйте работу программы скриншотами в отчете. Далее модифицируйте программу так, чтобы можно было вводить с клавиатуры начальное поколение. Например, ввести в цикле строки по количеству строк в консоли и в каждой позиции строки вводить либо звездочку (клетка живая), либо пробел – клетка мёртвая (см. скриншот ниже, цифрами отмечены номера строк при вводе).



Примерный код ввода начальной позиции (оформите самостоятельно в виде функци!):

```
for j in range(rows):
    print(f"{j}: ", end="")  # Выводим номер строки без перехода на новую строку
    str_input = input()      # Считываем строку ввода

    for i in range(min(cols, len(str_input))):
        if str_input[i] == '*':
            grid[i][j] = 1  # Если есть звездочка в строке, заносим в поле 1
        else: # иначе заносим 0
            grid[i][j] = 0
```

3. Игра «Змейка»

Описание программы: суть игры простая – змейка двигается по полю и собирает объекты (символы, яблоки), с каждым собранным символом длина змейки увеличивается и увеличивается количество очков. Цель – набрать как можно больше объектов. Змейка не должна выйти за границы экрана или упереться в саму себя (в свой хвост).

Импортируем библиотеки и определим основные константы и переменные:

```
# -*- coding: utf-8 -*-

import os
import random
import time
from threading import Thread

if os.getenv("OS") in ["Windows_NT", "Linux"]:
    import msvcrt as m

# Начальные значения
x = 5
y = 3
game_thread = True
fruit_cord_x = 5
fruit_cord_y = 6
button_default = "d"
score = 0
icon_player = "►"
tail = "o"
last2X, last2Y, lastX, lastY = 0, 0, 0, 0
elemX = [0] * 100
elemY = elemX.copy()
```

Функции очистки экрана консоли и завершения игры:

```
def clear():
    if os.getenv("OS") == "Windows_NT":
        os.system("cls")
    elif os.getenv("OS") in [None, "Linux"]:
        os.system("clear")


def game_over():
    global game_thread
    print("\nGAME OVER\nВаши очки: {0}".format(score))
    game_thread = False
    exit()
```

Напишем функцию, которая рисует игровое поле(доску-board):

```
def board(width: int = 40, height: int = 20, pos_player_x: int = x, pos_player_y:
int = y):
```

```
    global score, fruit_cord_x, fruit_cord_y, game_thread, icon_player, last2X,
lastX, lastY, last2Y, elemY, elemX
    clear()
    for i in range(height):
        for j in range(width):
            if pos_player_x == fruit_cord_x and pos_player_y == fruit_cord_y:
                fruit_cord_x = random.randint(5, width - 1)
                fruit_cord_y = random.randint(5, height - 1)
                score += 1

            for el in range(score):
                if pos_player_x == elemX[el] and pos_player_y == elemY[el]:
                    game_over()

            if not (x in range(width - 39)) and not (y in range(height - 1)) or
not (x in range(width - 1)) and not (
                    y in range(height - 19)):
                game_over()

            if j == 0:
                print("\t", end="")
                print('#', end='')
            elif i == 0:
                print('#', end='')
            elif i == height - 1:
                print('#', end='')
            elif j == width - 1:
                print('#', end='')
            elif pos_player_x == j and pos_player_y == i:
                print(icon_player, end='')
            elif fruit_cord_x == j and fruit_cord_y == i:
                print("*", end='')
            else:
                pr = True
                for ls in range(score):
                    if elemX[ls] == j and elemY[ls] == i:
                        print(tail, end="")
                        pr = False
                if pr:
                    print(' ', end='')

        print()
    # интерфейс
    print(f"\tОчки: {score}\n\n\t\tWASD / Стрелочки - перемещение\n\t\t\tESC -
выйти")
    lastX, lastY = pos_player_x, pos_player_y
    if score > 0:
        for el in range(score):
            last2X, last2Y = elemX[el], elemY[el]
            elemX[el], elemY[el] = lastX, lastY
            lastX, lastY = last2X, last2Y
```

Далее две функции работы с клавиатурой. Распознаются нажатия клавиш – стрелок управления курсором.

```
def button_move():
    global button_default
    if os.getenv("OS") in ["Windows_NT", "Linux"]:
        while game_thread:
            button_default = m.getch()[0]
    else:
        while game_thread:
            button_default = input("Нажмите кнопку перемещения: ")


def move():
    global x, y, game_thread, button_default, icon_player

    while game_thread:
        if button_default in ["", " "]:
            button_default = "d"
        elif button_default in ["w", 119, 230, 72]:
            y -= 1
            icon_player = "▲"
        elif button_default in ["a", 97, 228, 75]:
            x -= 1
            icon_player = "◄"
        elif button_default in ["s", 115, 235, 80]:
            y += 1
            icon_player = "▼"
        elif button_default in ["d", 100, 162, 77]:
            x += 1
            icon_player = "►"

        elif button_default in ["exit", 27]:
            print("Вы покинули игру\nВаши очки: {0} - выши очки не были
засчитаны".format(score))
            game_thread = False
            exit()

        board(pos_player_x=x, pos_player_y=y)

        time.sleep(.2)
```

Основная функция. В ней используются потоки (thread) для запуска функций работы с клавиатуры.

Модуль threading в Python позволяет реализовать многопоточность. Это означает, что можно выполнять несколько операций одновременно, используя разные потоки.

Некоторые ситуации, в которых может потребоваться использование многопоточности:

- Обработка нажатия кнопки в графическом/консольном интерфейсе. Если по нажатию кнопки требуется осуществлять множество действий, которые требуют времени,

соответствующие операции выполняются в другом потоке. Это необходимо для устранения вероятности «подвисания» графического/консольного интерфейса.
- Одновременное подключение нескольких устройств. Они могут быть подсоединены к разным COM-портам.
- Загрузка файлов из сети и одновременная обработка уже загруженных элементов.

Многопоточность позволяет улучшить производительность и ресурсоэффективность приложений.

Функция Thread() в Python позволяет создавать потоки с помощью модуля threading.

Синтаксис функции: variable = Thread(target=function_name, args=(arg1, arg2,)).

Первый аргумент (target) определяет «целевую» функцию, которая будет выполнена внутри потока. Далее следует список аргументов для этой функции. Если аргументы имеют фиксированное положение, то они передаются как args=(x, y). Если же требуются аргументы в виде пар «ключ-значение», можно использовать запись вида kwargs={'prop': 120}.

Для удобства отладки можно присвоить каждому новому потоку имя. Для этого нужно добавить параметр name="Имя потока" в параметры функции.

Функция start() используется для запуска созданного ранее потока.

После использования класса threading.Thread() появляется новый поток, но он неактивен. Чтобы запустить его в программном обеспечении, обязательно применение start-метода.

При вызове метода start() создаётся новый поток выполнения, в котором будет выполняться указанный код.

```
def main():
    board()
    Thread(target=move).start()
    Thread(target=button_move).start()
```

Определим функцию menu() для запуска игры:

```
def menu():
    clear()
    print("Программа консольная змейка | Управление - стрелки клавиатуры\n\n\t\t1
- Играть\n\t\t3 - Выход")
    if os.getenv("OS") in ["Windows_NT", "Linux"]:
        btn = m.getch()[0]
    else:
        btn = input("Выберите пункт: ")
    if btn in ["1", 49]:
        main()
    elif btn in ["3", 51]:
        exit()
```

Ну и собственно сама программа состоит из одно строчки – запуска функции menu()

```
menu()
```

ЗАДАНИЕ. Отладьте код программы «Змейка», зафиксируйте работу программы скриншотами в отчете. Далее модифицируйте программу так:

1) Чтобы можно было вводить в функции menu перед игрой имя пользователя, а затем в конце игры записывать его имя и результат (а можно еще и дату-время) в файл.
2) Добавить в меню пункт – «2. Посмотреть результаты». Тогда программа считывает файл результатов и выдает на экран.
3) **По желанию:** попробуйте реализовать игру без многопоточности, сравните результаты и сделайте выводы.

Рекомендации по пп.1-2. Вот так, например, могут выглядеть функции работы с файлами в формате json:

```
import json

def read_file():
    file = open('settings.txt', 'r+')
    read = file.read()
    return json.loads(read)


def write_file(name: str, task_id: int = 0):
    file = open('settings.txt', 'w+')
    file.write('{"name": "'+name+'", "task_id": '+str(task_id)+'}')
    return True
```

А вот так, например, может выглядеть ФРАГМЕНТ основной программы:

```
try:
        read_file()
except Exception as err:
        status_auth = False
        while not status_auth:
            clear()
            nickname = input("Система не обнаружила файл настроек\n\nПридумайте
себе ник: ")
            if nickname not in ["", " "] and len(nickname) != 0:
                status_auth = True
            else:
                print("ERROR: Поле осталось пустым. Исправьте это!")
                time.sleep(1.5)

menu()
```