ҚазақстанРеспубликасыОқу-ағартуминистрлігі

«Білім» кәсіби гуманитарлық-техникалық колледжі



**Зертханалық жұмыс №1**

**Тақырыбы: ШарттардыпрограммалаужәнеPyGameкітапханасынқосу**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Жұмыстың орындау сапасы | Баға диапазоны | Орындаған % |
| 1 | Орындалған жоқ, сабақта себепсіз болмады. | 0 % |  |
| 2 | Жұмыстың орындалуы және студенттің белсенділігі | 0-50% |  |
| 3 | Жұмысты рәсімдеу | 0-20% |  |
| 4 | Анықтамалар мен техникалық әдістемелерді, пәннің оқу-әдістемелік кешенін, лекция конспектілерін қолдана білу. | 0-5% |  |
| 5 | Техникалық құралдарды пайдалана білу | 0-5% |  |
| 6 | Жұмысты қорғау | 0-20% |  |
|  | Қорытынды | 0-100% |  |

Оқытушы: Нургисаева У.М.

Студент: Бегжан Бексұлтан

Тобы: 3БҚ-1-22

Тараз 2025

**Зертханалықжұмыс №1**

**Тақырыбы**: ШарттардыпрограммалаужәнеPyGameкітапханасынқосу

**Мақсаты:**Студенттергешарттыоператорлар мен логикалықоператорлардықолданудыүйрету.**PyGame**кітапханасыарқылыойындаржасаунегіздерінмеңгерту.Шарттыоператорлар мен PyGame-дыбіріктіруарқылықарапайымойынлогикасынқұрудыүйрету.

**Міндеттері:**

* PyGameкітапханасынорнатужәнеконфигурациялау:
* Қарапайымойынлогикасынқұру:
* Қарапайыманимациялардыжасау:
* Қателерменжұмысістеу:

**Қажеттіқұралдар:**

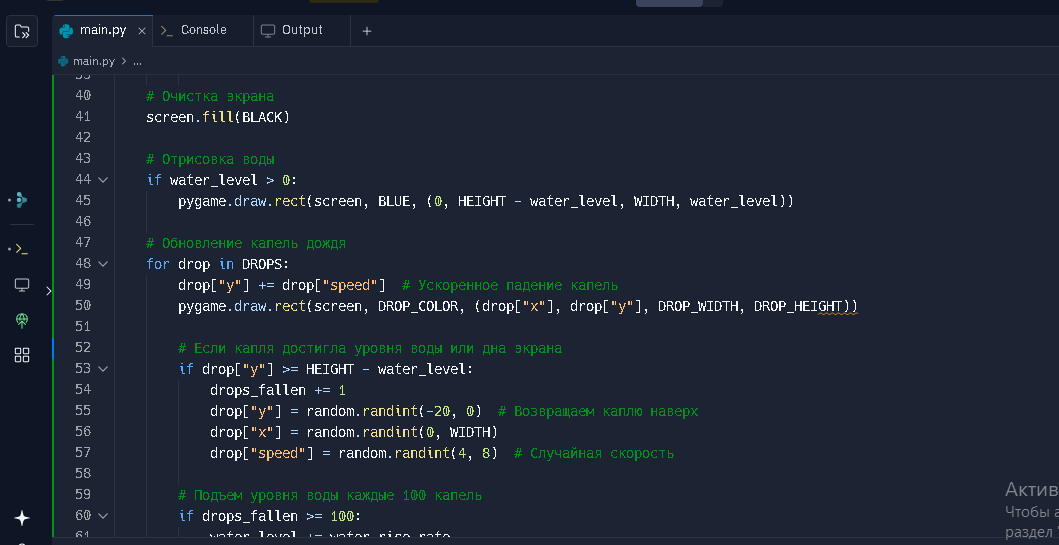
1. Бағдарламалауортасы: VS Code
2. Дербес компьютер немесе ноутбук

**1-Тапсырма-Жаңбыр**

Pygameкөмегіменжаңбыртренажерінжазыңыз: әржүзтамшыныңқұлауы су деңгейінің 1 пиксельгекөтерілуінеәкеледі.

**Задание 1 – Дождь**

Используя Pygame, напишите симулятор дождя: падение каждой сотни капель приводит к подъему уровня воды на 1 пиксель.



**Разбор кода пошагово:**

**Инициализация Pygame**

 import pygame — подключает библиотеку Pygame для работы с графикой.

 import random — импортирует модуль для генерации случайных чисел (используется для случайного расположения и скорости капель).

 pygame.init() — инициализирует все модули Pygame.

**Настройки окна**

 Устанавливаются размеры окна 800x600.

 pygame.display.set\_mode((WIDTH, HEIGHT)) — создаёт окно.

 pygame.display.set\_caption("Симулятор дождя") — задаёт название окна.

**Цвета**

 BLACK — цвет фона.

 BLUE — цвет воды.

 LIGHT\_BLUE — цвет капель дождя.

**Настройки капель дождя**

 NUM\_DROPS = 200 — количество капель дождя.

 DROPS = [...] — создаётся список капель, каждая из которых представлена словарём:

* "x" — случайная горизонтальная позиция.
* "y" — случайная вертикальная позиция (от -HEIGHT до 0, чтобы капли появлялись сверху).
* "speed" — случайная скорость (от 4 до 8 пикселей за кадр).

 DROP\_COLOR — цвет капель.

 DROP\_WIDTH, DROP\_HEIGHT — размеры капель (2x10 пикселей).

**Переменные для воды**

* water\_level — текущий уровень воды.
* water\_rise\_rate — на сколько пикселей поднимается вода после 100 упавших капель.
* drops\_fallen — счётчик капель, достигших дна.

**FPS (Кадры в секунду)**

 pygame.time.Clock() — создаёт объект для управления частотой кадров.

 FPS = 60 — ограничение кадров в секунду (для плавности анимации).

**Главный цикл игры**

 while running: — основной игровой цикл, который будет выполняться, пока не нажмут кнопку "Закрыть".

 Обрабатываются события:

* Если игрок нажмёт Закрыть (pygame.QUIT), программа завершит работу.

**Очистка экрана и рисование воды**

 screen.fill(BLACK) — очищает экран, заполняя его черным цветом.

 Рисуется вода:

* if water\_level > 0: — если уровень воды больше 0, рисуется синий прямоугольник внизу экрана.
* pygame.draw.rect(screen, BLUE, (0, HEIGHT - water\_level, WIDTH, water\_level)) — создаёт "водоём", который поднимается снизу вверх.

**Движение капель дождя**

 Перебираем каждую каплю в списке DROPS.

 drop["y"] += drop["speed"] — капля опускается вниз на свою скорость.

 pygame.draw.rect(...) — отрисовка капли.

**Обработка падения капель**

Если капля достигла воды или низа экрана (drop["y"] >= HEIGHT - water\_level):

1. Счётчик капель (drops\_fallen) увеличивается.
2. Капля перезапускается сверху (y становится случайным значением от -20 до 0).
3. Позиция x и скорость обновляются случайно.

**Подъём уровня воды**

Когда 100 капель достигли воды:

* Уровень воды поднимается на water\_rise\_rate (2 пикселя).
* Счётчик drops\_fallen сбрасывается в 0.

**Ограничение уровня воды**

Чтобы вода не вышла за пределы экрана, если она достигнет HEIGHT, её значение фиксируется.

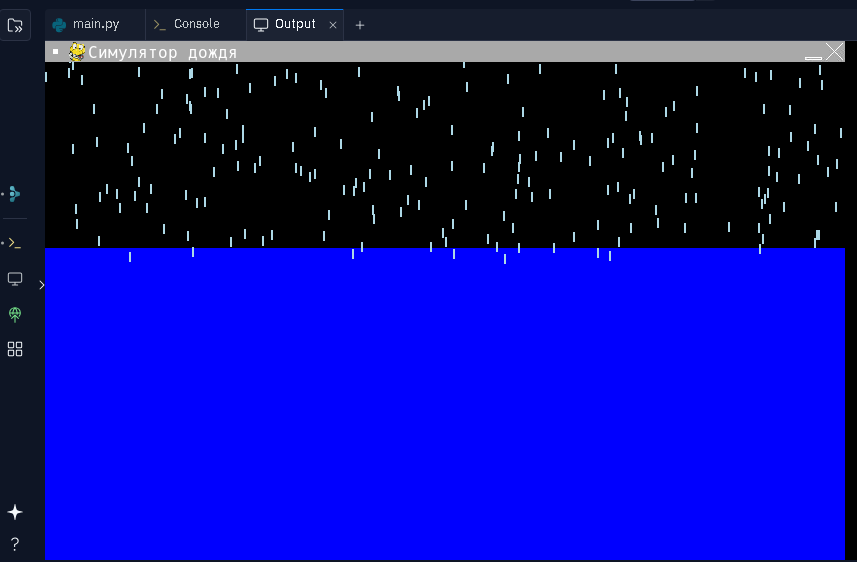
**Обновление экрана и задержка**

 pygame.display.flip() — обновляет экран, чтобы показать все изменения.

 clock.tick(FPS) — ограничивает частоту кадров до 60 FPS.

Завершение работы

pygame.quit() — закрывает окно и завершает работу Pygame.

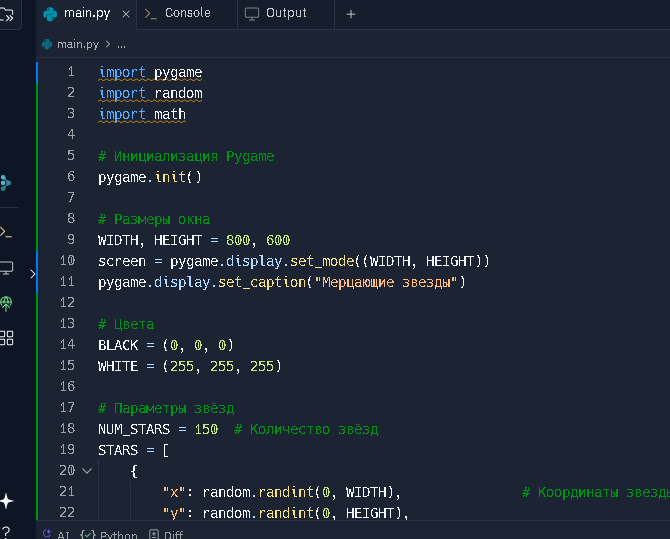


**2-Тапсырма - Жарқырайтынжұлдыздар**

Pygameкөмегіменжұлдыздыаспантренажерінжазыңыз-жұлдыздарболыптабылатыншеңберлержыпылықтаудыимитациялауүшінкішірейіп, кеңейеді.

**Задание 2 – Мерцающие звезды**

Используя Pygame, напишите симулятор звездного неба – окружности, представляющие собой звезды, сжимаются и расширяются, имитируя мерцание.



**Разбор кода "Мерцающие звёзды" пошагово**

**Импорт библиотек**

import pygame

import random

import math

* pygame — используется для графики и анимации.
* random — для случайных значений (координаты, радиус звёзд, скорость мерцания).
* math — нужен для использования синуса, чтобы делать эффект мерцания.

**Инициализация Pygame**

pygame.init()

* Запускает Pygame, чтобы можно было работать с графикой и событиями.

**Настройки окна**

WIDTH, HEIGHT = 800, 600

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Мерцающие звезды")

* WIDTH, HEIGHT = 800, 600 — размеры окна.
* pygame.display.set\_mode((WIDTH, HEIGHT)) — создаёт окно Pygame.
* pygame.display.set\_caption("Мерцающие звезды") — задаёт заголовок окна.

**Цвета**

BLACK = (0, 0, 0) # Фон

WHITE = (255, 255, 255) # Звёзды

* BLACK — чёрный цвет (фон).
* WHITE — белый цвет (звёзды).

**Создание звёзд**

NUM\_STARS = 150 # Количество звёзд

STARS = [

{

"x": random.randint(0, WIDTH), # Координаты звезды

"y": random.randint(0, HEIGHT),

"base\_radius": random.randint(2, 4), # Базовый радиус звезды

"radius": 0, # Текущий радиус (будет меняться)

"speed": random.uniform(0.02, 0.05), # Скорость мерцания (разная у каждой)

"phase": random.uniform(0, 2 \* math.pi), # Фаза (чтобы звёзды начинали мерцать в разное время)

}

for \_ in range(NUM\_STARS)

]

Каждая звезда представлена словарём со следующими параметрами:

* "x", "y" — случайное положение звезды.
* "base\_radius" — базовый размер (слегка различается у каждой звезды).
* "radius" — текущий размер (он будет изменяться).
* "speed" — скорость мерцания (разная для каждой звезды).
* "phase" — начальная фаза мерцания, чтобы все звёзды не мерцали синхронно.

Создаётся список STARS из 150 таких звёзд.

**Настройки FPS**

clock = pygame.time.Clock()

FPS = 60

* clock = pygame.time.Clock() — создаёт объект для управления частотой кадров.
* FPS = 60 — ограничение FPS до 60 кадров в секунду.

**Главный цикл**

running = True

while running:

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

* Бесконечный цикл while running — выполняется, пока окно не закроют.
* Обрабатываются события:
  + Если пользователь нажмёт "Закрыть", то running = False, и программа завершится.

**Очистка экрана**

screen.fill(BLACK)

* Каждый кадр экран очищается чёрным цветом, чтобы не оставались следы звёзд.

**Обновление и отрисовка звёзд**

for star in STARS:

# Анимация мерцания (радиус меняется плавно с синусом)

star["radius"] = star["base\_radius"] + math.sin(pygame.time.get\_ticks() \* star["speed"] + star["phase"]) \* 2

# Отрисовка звезды

pygame.draw.circle(screen, WHITE, (star["x"], star["y"]), max(1, int(star["radius"])))

🔹 Как работает мерцание?

* math.sin(...) даёт плавное изменение радиуса от base\_radius - 2 до base\_radius + 2.
* pygame.time.get\_ticks() — время в миллисекундах (увеличивается постоянно).
* star["speed"] регулирует скорость мерцания.
* star["phase"] даёт случайный сдвиг, чтобы звёзды не мерцали одновременно.
* pygame.draw.circle(...) рисует звезду:
  + Цвет: WHITE
  + Координаты: (star["x"], star["y"])
  + Радиус: max(1, int(star["radius"])) (чтобы не было отрицательных значений).

**Обновление экрана и задержка**

pygame.display.flip()

clock.tick(FPS)

* pygame.display.flip() — обновляет экран, чтобы показать изменения.
* clock.tick(FPS) — ограничивает частоту кадров до 60 FPS.

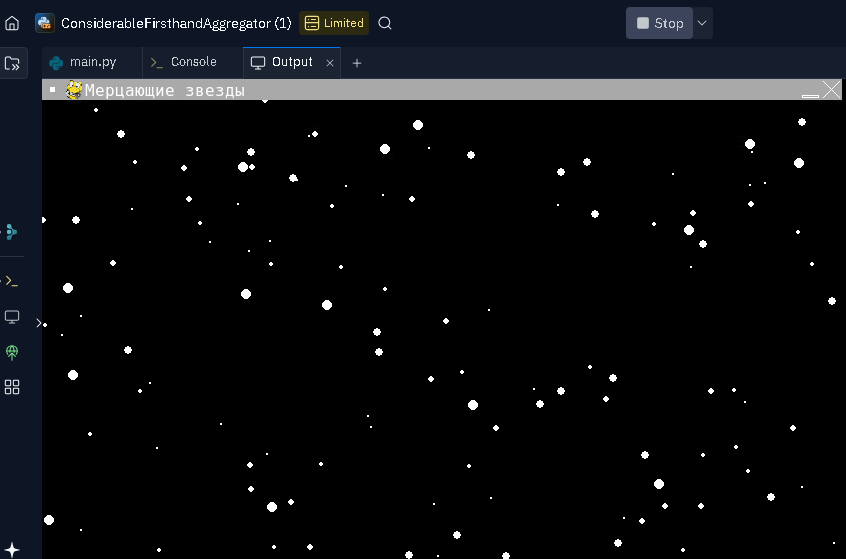
**Завершение программы**

pygame.quit()

* Закрывает Pygame и освобождает ресурсы.

🎯 Что делает код?

✅ Создаёт звёздное небо с 150 звёздами.  
✅ Каждая звезда плавно мерцает с разной скоростью.  
✅ Мерцание выглядит естественно, потому что звёзды мерцают не одновременно.  
✅ Работает плавно (60 FPS).



**3-Тапсырма - Колобок**

Pygameкөмегіментүлкі (осы кадрлардантұратын) колобоктықуыпжететін анимация жасаңыз. Колобок өзосініңайналасындаайналады.

**Задание 3 – Колобок**

Используя Pygame, создайте анимацию, в которой лиса ([состоящая из этих фреймов](https://github.com/natkaida/pygame_projects/tree/main/zadanie_5_kolobok/assets)) преследует [Колобка](https://github.com/natkaida/pygame_projects/blob/main/zadanie_5_kolobok/assets/kolobok.png). Колобок вращается вокруг своей оси.

**Разбор кода "Колобок и Лиса" пошагово**

**Импорт библиотеки Pygame**

import pygame

pygame.init()

* Импортируется pygame для работы с графикой и анимацией.
* pygame.init() — инициализирует библиотеку.

**Настройки окна**

background = (24, 113, 147) # Цвет фона

WINDOW\_WIDTH = 800

WINDOW\_HEIGHT = 300

game\_display = pygame.display.set\_mode((WINDOW\_WIDTH, WINDOW\_HEIGHT))

pygame.display.set\_caption('Колобок')

* Задаём размер окна: 800x300 пикселей.
* Цвет фона: тёмно-голубой (24, 113, 147).
* Создаём окно: pygame.display.set\_mode(...).
* Устанавливаем заголовок: "Колобок".

**Загрузка изображения Колобка**

kolobok = pygame.image.load('kolobok.png')

* Загружается картинка "Колобок" (kolobok.png).

**Настройки вращения Колобка**

kolobok\_angle = 0 # Начальный угол

kolobok\_rotation\_speed = 2 # Скорость вращения

* Колобок будет вращаться на 2° за каждый кадр.

**Загрузка фреймов лисы (анимация)**

fox = []

for i in range(8):

fox.append(pygame.image.load(f'fox{i+1}.png'))

* Лиса анимирована и состоит из 8 кадров (fox1.png, fox2.png ... fox8.png).
* Загружаем все фреймы в список fox.

**Настройка анимации лисы**

fox\_frame = 0 # Текущий кадр

fox\_frame\_rate = 8 # Скорость анимации (кадров в секунду)

fox\_frame\_timer = 0 # Таймер для смены кадров

* Лиса меняет кадры 8 раз в секунду.
* Начинаем с первого кадра (fox\_frame = 0).
* fox\_frame\_timer отслеживает время для смены кадра.

**Стартовые позиции Колобка и Лисы**

kolobok\_x = 0 # Колобок стартует слева

kolobok\_y = WINDOW\_HEIGHT // 2 + kolobok.get\_height() // 4 # Центр окна

fox\_x = -fox[0].get\_width() # Лиса появляется за левым краем

fox\_y = WINDOW\_HEIGHT // 2 - fox[0].get\_height() // 2 # Лиса в центре по высоте

movement\_speed = 3 # Скорость движения

* Колобок и Лиса начинают движение слева направо.
* Лиса изначально невидима (находится за левым краем экрана).
* Они двигаются со скоростью 3 пикселя за кадр.

**Настройка FPS**

clock = pygame.time.Clock()

* Создаём объект Clock для контроля FPS.

**Главный цикл игры**

game\_exit = False

while not game\_exit:

for event in pygame.event.get():

if event.type == pygame.QUIT:

game\_exit = True

* Цикл while работает, пока game\_exit = False.
* Если игрок нажмёт "Закрыть окно", игра завершится.

**Вращение Колобка**

kolobok\_angle += kolobok\_rotation\_speed

if kolobok\_angle >= 360:

kolobok\_angle = 0

rotated\_kolobok = pygame.transform.rotate(kolobok, kolobok\_angle)

* Угол увеличивается на 2° за кадр.
* Когда угол достигает 360°, он сбрасывается в 0.
* pygame.transform.rotate() поворачивает изображение.

**Движение Колобка и Лисы**

kolobok\_x += movement\_speed

if kolobok\_x > WINDOW\_WIDTH:

kolobok\_x = 0 - fox[0].get\_width()

fox\_x += movement\_speed

if fox\_x > WINDOW\_WIDTH:

fox\_x = 0 - fox[0].get\_width()

* Оба персонажа движутся слева направо.
* Когда Колобок или Лиса выходят за экран, они появляются снова слева.

**Анимация Лисы (смена кадров)**

fox\_frame\_timer += clock.tick(60)

if fox\_frame\_timer >= 1000 / fox\_frame\_rate:

fox\_frame\_timer -= 1000 / fox\_frame\_rate

fox\_frame = (fox\_frame + 1) % len(fox)

* fox\_frame\_timer накапливает время, прошедшее с прошлого кадра.
* Когда проходит 1000 / fox\_frame\_rate миллисекунд (примерно 125 мс), происходит смена кадра.
* Фрейм переключается циклично (% len(fox)).

**Отрисовка сцены**

game\_display.fill(background) # Заполняем фон

game\_display.blit(rotated\_kolobok, (kolobok\_x, kolobok\_y)) # Рисуем Колобка

game\_display.blit(fox[fox\_frame], (fox\_x, fox\_y)) # Рисуем анимацию Лисы

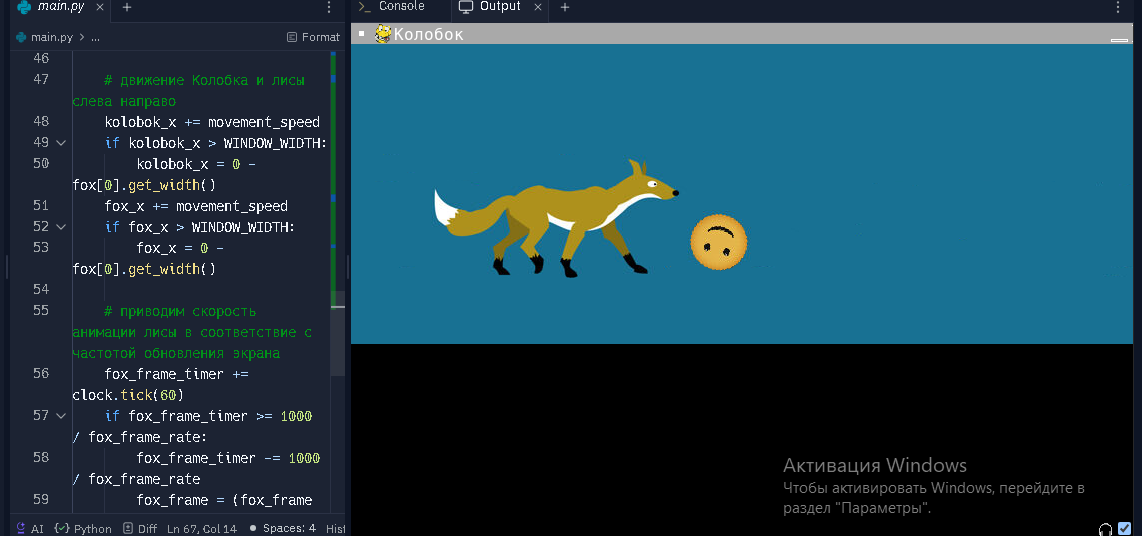
pygame.display.update() # Обновляем экран

* Очищаем экран цветом background.
* Рисуем Колобка (повёрнутый).
* Рисуем Лису (анимированную).
* Обновляем экран (pygame.display.update()).

**Завершение программы**

pygame.quit()

* Закрывает Pygame и освобождает ресурсы.



**4-Тапсырма - Визуалды жад**

Pygameмүмкіндіктерінпайдаланып Memory gameойыныныңжеңілнұсқасынжазыңыз. Алдыменбағдарламатүрлі-түстішеңберлердішығарады (кездейсоқретпен) жәнепайдаланушығаолардыңорналасқанжерінбірнеше секунд ішіндеестесақтауғамүмкіндікбереді. Соданкейінбағдарламатүрлі-түстішеңберлердісұртүспенжабады: пайдаланушы жад бойыншатүрлі-түстіжұптардысәйкестендіруі керек. Әрбірболжамдыжұппайдаланушыға 1 ұпайбереді.

**Задание 4 – Визуальная память**

Напишите лайт-версию игры Memory game, используя возможности Pygame. Сначала приложение выводит (в случайном порядке) цветные окружности и дает возможность пользователю запомнить их расположение в течение нескольких секунд. Затем приложение закрывает цветные окружности серыми: пользователь должен по памяти сопоставить цветные пары. Каждая угаданная пара приносит пользователю 1 балл.

Этот код — игра на тренировку визуальной памяти, где игрок должен запомнить цвета окружностей и затем находить одинаковые пары. Давай разберём его по шагам.

**1. Инициализация Pygame и базовые настройки**

import pygame

from random import shuffle

pygame.init()

# Определяем цвета

black = (0, 0, 0)

white = (255, 255, 255)

red = (255, 0, 0)

blue = (0, 0, 255)

green = (0, 255, 0)

yellow = (255, 255, 0)

purple = (128, 0, 128)

grey = (192, 192, 192)

# Размер окна

screen\_width = 800

screen\_height = 600

screen = pygame.display.set\_mode((screen\_width, screen\_height))

pygame.display.set\_caption("Тренируем визуальную память")

* Здесь мы импортируем pygame и shuffle для перемешивания пар цветов.
* Определяем размеры окна (800x600) и основные цвета.
* Создаём окно и задаём его заголовок.

**2. Генерация окружностей и перемешивание пар**

# Радиус окружности

circle\_radius = 50

# 6 цветов, по 2 каждой (всего 12 кругов)

circle\_colors = [red, blue, green, yellow, purple, white]

circle\_pairs = circle\_colors \* 2

shuffle(circle\_pairs)

* У нас 6 цветов, и мы создаём 12 окружностей (по 2 одинаковых цвета).
* shuffle(circle\_pairs) перемешивает их случайным образом.

**3. Распределение окружностей на экране**

circle\_positions = []

for i in range(6):

for j in range(2):

center\_x = ((screen\_width / 6) \* (i + 1)) - (screen\_width / 12)

center\_y = ((screen\_height / 3) \* (j + 1)) - (screen\_height / 6)

circle\_positions.append([center\_x, center\_y])

* Мы размещаем 12 окружностей в 2 строки по 6 кругов.
* Центр каждого круга рассчитывается с равными промежутками.

**4. Сохранение позиций и цветов окружностей**

original\_circle\_positions = circle\_positions.copy()

original\_circle\_colors = circle\_pairs.copy()

* Мы копируем позиции и цвета, чтобы потом сверять их при открытии.

**5. Отображение цветных окружностей**

for i in range(len(circle\_pairs)):

position = circle\_positions[i]

color = circle\_pairs[i]

pygame.draw.circle(screen, color, position, circle\_radius)

pygame.display.update()

* Отрисовываем все 12 окружностей.
* Затем обновляем экран, чтобы они появились.

**6. Задержка перед скрытием цветов**

pygame.time.wait(5000)

* Игроку даётся **5 секунд**, чтобы запомнить цвета.

**7. Закрытие цветов серыми окружностями**

for i in range(len(circle\_pairs)):

position = circle\_positions[i]

pygame.draw.circle(screen, grey, position, circle\_radius)

pygame.display.update()

* После 5 секунд все круги закрашиваются серым цветом.

**8. Логика взаимодействия с игроком**

uncovered\_circles = []

last\_uncovered\_circle = None

score = 0

* uncovered\_circles — список открытых пользователем окружностей.
* last\_uncovered\_circle — запоминает последний открытый круг.
* score — счётчик правильных совпадений.

**9. Основной игровой цикл**

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

exit()

* Игровой цикл работает постоянно, пока пользователь не закроет окно.

**10. Определение нажатия на окружности**

if event.type == pygame.MOUSEBUTTONDOWN:

mouse\_pos = event.pos

for i in range(len(circle\_positions)):

position = circle\_positions[i]

if ((position[0] - mouse\_pos[0]) \*\* 2 + (position[1] - mouse\_pos[1]) \*\* 2) \*\* 0.5 < circle\_radius:

if i not in uncovered\_circles:

uncovered\_circles.append(i)

color = original\_circle\_colors[i]

pygame.draw.circle(screen, color, position, circle\_radius)

pygame.display.update()

* Проверяем, на какую окружность нажал пользователь.
* Если она ещё не открыта, добавляем её в uncovered\_circles.
* Открываем её (рисуем цветную окружность).

**11. Проверка на совпадение цветов**

if last\_uncovered\_circle is not None and original\_circle\_colors[last\_uncovered\_circle] == original\_circle\_colors[i]:

score += 1

last\_uncovered\_circle = i

* Если текущая и предыдущая окружность совпадают по цвету, увеличиваем score.
* Запоминаем последнюю открытую окружность.

**12. Завершение игры**

if len(uncovered\_circles) == len(circle\_pairs):

# Вывод результата

final\_score\_text = font.render(f"Уровень памяти: {str(score)} из 6", True, white)

screen.blit(final\_score\_text, (screen\_width // 2, screen\_height // 2 + 125))

pygame.display.update()

pygame.time.wait(3000)

pygame.quit()

exit()

* Если игрок открыл все окружности, показываем результат.
* Выводим текст с количеством найденных пар.
* Ждём 3 секунды, а затем закрываем игру.

