**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**БИШКЕКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ им. К. КАРАСАЕВА**



**КУРСОВАЯ РАБОТА**

**По дисциплине**: Дисциплина профиля по выбору студента

**Тема:**« Простая игра "Змейка" на Python (Pygame)»

**Выполнила:** Асыранкулова Алия

студентка 3 курса

**Научный руководитель:** Жакыпова Н. Н.

**Бишкек 2025**

**Оглавление**

[Введение 3](#_Toc190860002)

[1.Теоретическая часть. 4](#_Toc190860003)

[**1.1. Описание проекта** 4](#_Toc190860004)

[**1.2. Используемые технологии** 5](#_Toc190860005)

[2. Практическая часть 8](#_Toc190860006)

[**2.1. Анализ требований** 8](#_Toc190860007)

[**2.2. Проектирование интерфейса** 10](#_Toc190860008)

[3. Реализация 12](#_Toc190860009)

[**3.1. Разработка логики игры** 13](#_Toc190860010)

[**3.2. Разработка графики и анимации** 15](#_Toc190860011)

[**3.3. Реализация игрового интерфейса** 18](#_Toc190860012)

[4. Тестирование 23](#_Toc190860013)

[5. Размещение в GitHub 25](#_Toc190860014)

[Заключение 26](#_Toc190860015)

[Список литературы 30](#_Toc190860016)

[Листинг 31](#_Toc190860017)

# **Введение**

В данной курсовой работе рассматривается процесс разработки классической игры 'Змейка' с использованием языка программирования Python. Игрок управляет змейкой, которая растет с каждым съеденным предметом. Описывается логика игры, включая создание игрового поля, управление движением змейки, а также обработку коллизий. Работа включает различные методы и подходы, которые могут использоваться как новичками, так и более опытными программистами. Анализируются особенности работы с библиотеками, такими как Pygame, для создания графического интерфейса и управления игровыми элементами. В результате, представляется окончательная версия игры, которую можно запустить и протестировать на различных устройствах.

Цель проекта — Целью данной работы является создание игры 'Змейка' на языке Python, что позволит приобрести навыки программирования и разработки игр, а также повысить уровень знаний в области использования библиотек для работы с графикой.

# **1.Теоретическая часть.**

## **1.1. Описание проекта**

Проект представляет собой веб-приложение, которое выполняет следующие функции:

* Управление змейкой.
* Появление пищи.
* Увеличение длины змейки.
* Проверка на столкновение.
* Подсчёт очков.

Приложение состоит из **клиентской части (frontend)**

**Frontend** отвечает за отображение веб-страниц, а также за ввод и вывод данных. В данном случае клиентская часть — это сам процесс игры, реализованный с использованием **Pygame**. Эта часть отвечает за:

* Отображение игрового поля, змейки, фруктов и других элементов интерфейса.
* Обработку ввода пользователя (например, движения змейки с помощью клавиш).

Функционал системы:

* Управление движением змейки: Игрок может управлять направлением змейки с помощью клавиш(вверх,вниз,влево,направо).
* Появление пищи: Каждый раз, когда змейка съедает пищу, она становится длиннее, и на поле пища появляется в разных местах.
* Проверка на столкновение: Если змейка выходит за границы игрового поля или сталкивается собой то игра заканчивается.

## **1.2. Используемые технологии**

Для разработки классической игры “Змейка” были использованы современные веб-технологии, каждая из которых выполняла определенную роль в проекте.

**Python**– разработка серверной части.

**Pygame-** библиотека Python для создания 2D-игр.

**Git, GitHub** – контроль версий и размещение кода.

**Python** — это высокоуровневый язык программирования, который был создан Гвидо Ван Россумом и впервые выпущен в 1991 году. Он известен своей простотой и читаемостью, что делает его идеальным для начинающих программистов. Python поддерживает несколько парадигм программирования, включая объектно-ориентированное, процедурное и функциональное программирование. Это означает, что вы можете использовать Python для решения самых разнообразных задач, от написания простых скриптов до разработки сложных приложений.

**Характеристики языка:**

* Динамическая типизация (не нужно указывать типы переменных).
* Высокий уровень абстракции (не требует работы с памятью).
* Большая стандартная библиотека (поддержка работы с файлами, сетями, базами данных и т. д.).

**Реализация в проекте :**

* **Игровая логика**: Python используется для разработки основной логики игры: движение змейки, обработка столкновений,проверка условий победы и поражения также добавление блоков в змейку.
* **Обработка ввода с клавиатуры**: ввод с клавиатуры, чтобы могу управлять змейкой ,используя клавиши для изменения направления движения
* **Отображение графики**: создание анимации, отображение графических объектов.
* **Звуковые эффекты**:с помощью pygame.mixer можно воспроизводить звуковые эффекты .
* **Структуры данных**:используется список для хранения тела змейки,а также векторы для позиционирования объектов.
* **Реализация классов**: Таких как Snake,Fruit,Main.

**Pygame** — это «игровая библиотека», набор инструментов, помогающих программистам создавать игры. К ним относятся:

* Графика и анимация
* Звук (включая музыку)
* Управление (мышь, клавиатура, геймпад и так далее)

**Характеристики:**

* **Обработка ввода (события).**
* **Обновление игры**.
* **Рендеринг (прорисовка).**

**Реализация в проекте:**

* **Обработка столкновений проверка столкновения с границами или ее собственным телом**.
* **Отображение графики** для отображения головы, хвоста и тела змейки также для еды.
* **Основной игровой цикл**  обрабатываются события (ввод пользователя, обновление игры),обновляется экран, продолжается до тех пор пока не произойдет завершение.
* **Обработка ввода от пользователя** такие как нажатие клавиш,чтобы управлять змейкой.
* **Инициализация\_Pygame** создается окно для игры с pygame.dispaly.set\_mode(). Устанавливается размер экрана.

# **2. Практическая часть**

## **2.1. Анализ требований**

#### **Функциональные требования**:

1. **Запуск и управление игрой**:Игрок должен иметь возможность начать игру, нажав кнопку "Начать игру". Управление змейкой осуществляется с помощью клавиш W, A, S, D.
2. **Игровая механика**: Змейка увеличивается при поедании пищи. Система автоматически отслеживает столкновения с границами экрана и с телом змейки.
3. **Графика и анимация**: Необходимо визуализировать змейку и пищу. Предполагается использование графических изображений для различных частей тела змейки и для еды (например, яблоки или другие объекты).
4. **Звуковые эффекты**: При поедании пищи должен воспроизводиться один звуковой эффект, а при завершении игры — другой.
5. **Счет и игровой процесс**: Система должна отображать количество очков в виде длины змейки, которая увеличивается с каждым съеденным объектом.

#### **Нефункциональные требования**:

1. **Производительность**: Игра должна функционировать без задержек или лагов, даже если змейка достигла значительных размеров и на экране присутствует множество объектов.
2. **Безопасность**: В данном контексте требования к безопасности минимальны, так как игра не использует персональные данные игроков. Однако, если бы была разработана версия с возможностью сохранения данных пользователей, необходима была бы защита этих данных.
3. **Масштабируемость**: Игра должна быть легко адаптируема к различным размерам экранов и иметь возможность изменения разрешения.
4. **Удобство интерфейса**: Игра должна быть интуитивно понятной и легкой в использовании, иметь четкие инструкции для игроков и удобные элементы управления (кнопки "Начать игру").

## **2.2. Проектирование интерфейса**

Интерфейс игры "Змейка" состоит из нескольких экранов и элементов управления, которые обеспечивают удобство использования и интуитивно понятный геймплей.

#### **1. Главный экран**

**Цель**: Экран для начала игры, отображающий кнопку "Начать игру" и отображение счета.

**Элементы интерфейса**:

* **Кнопка “Начать игру”** – запускает новую игру.
* **Фон** – графика, которая символизирует поле для игры (можно использовать текстуры травы или сетки).

#### **2. Экран игры (Игровое поле)**

**Цель**: Экран, на котором происходит сам игровой процесс.

**Элементы интерфейса**:

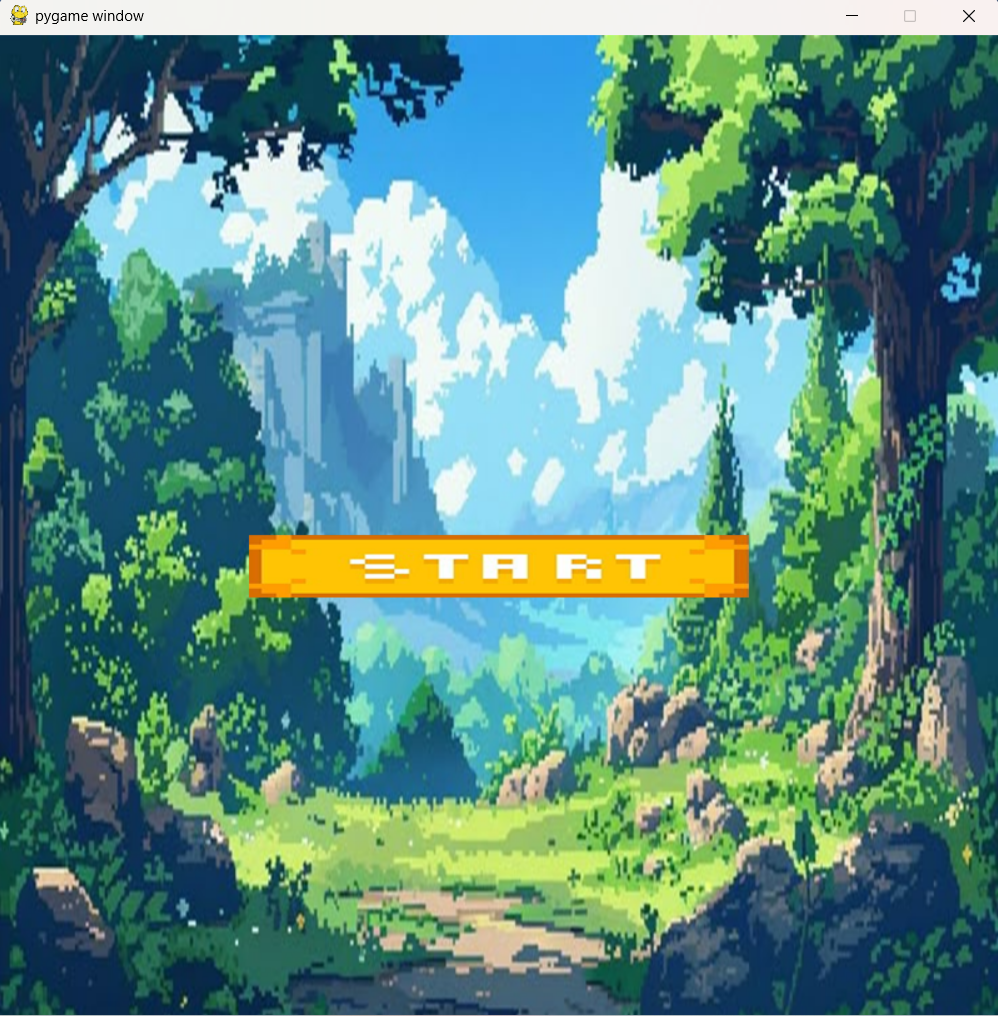
* + **Игровое поле** – отображается сетка, на которой движется змейка.
  + **Змейка** – управляемый объект, который растет по мере съедания пищи.
  + **Пища (виноград)** – объекты, которые змейка должна съедать для увеличения длины.

**3.Конец игры(завершающий экран)**

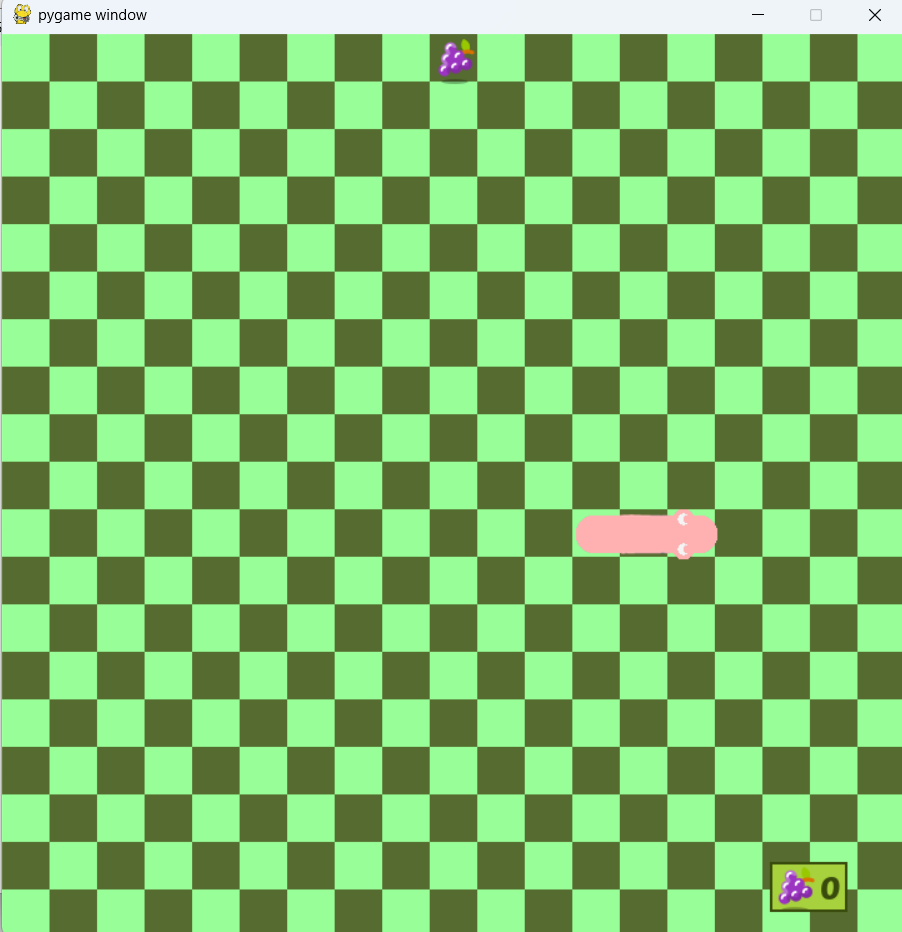
**Цель**: Экран для завершения игры, отображающий кнопку "Начать игру снова" и отображение счета.

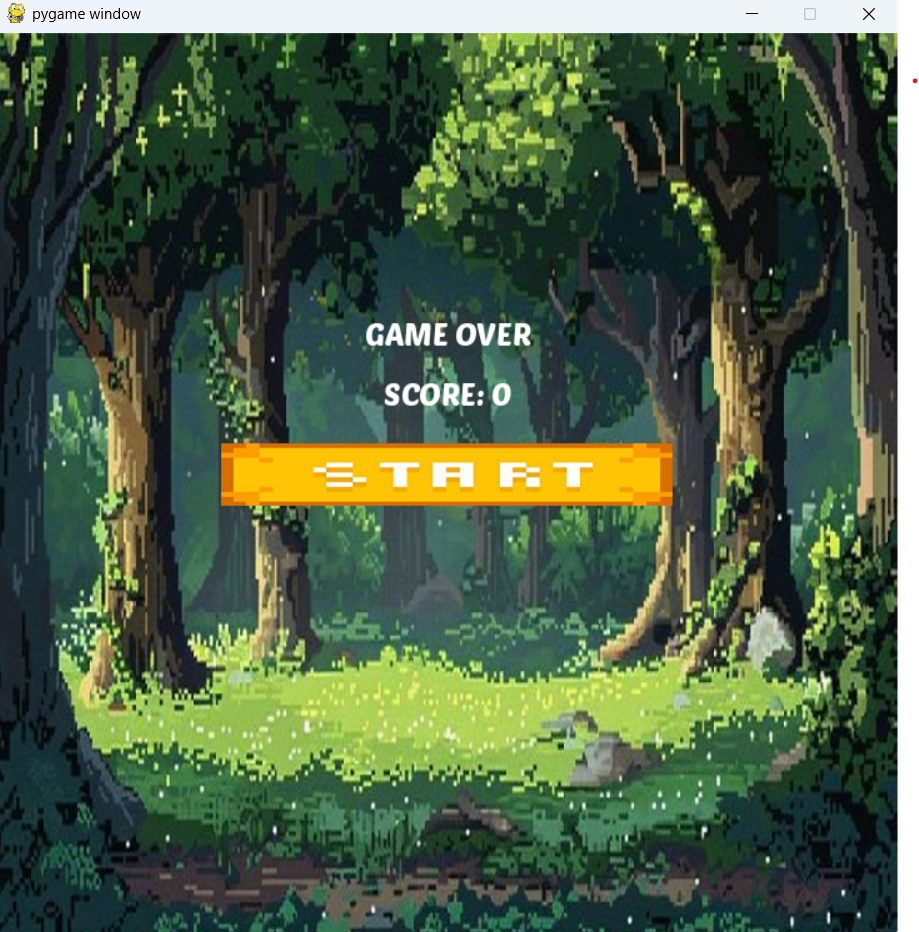
* **Кнопка “Начать игру заново”-**запускает игру снова.

**Пример структуры интерфейса:**

📌 **Главный экран**

📌 **Экран игры (Игровое поле)**

****

📌 **Конец игры (Завершающий экран)**

# **3. Реализация**

На этапе реализации создается сам программный продукт: пишется код, создается игровая логика, соединяются интерфейсная и игровая части.

В проекте использовано:

* **Фронтенд: Python** (Pygame)

**1. Игровая логика**

Для реализации основной логики игры был использован **Python** с библиотекой **Pygame**. Основные элементы игры, такие как змейка, фрукты и поле, взаимодействуют друг с другом. Змейка управляется игроком с помощью клавиш, и её движение обновляется каждый раз, когда происходит игровой цикл. Логика игры включает в себя:

* Обработку нажатий клавиш для управления движением змейки.
* Добавление нового блока к телу змейки при поедании фрукта.
* Проверку столкновений с границами поля или с самим телом змейки.
* Отображение экрана "Game Over" при завершении игры.

**2. Интеграция интерфейса**

Интерфейс игры был создан с использованием **Pygame** для отображения всех элементов на экране, включая:

* Игровое поле, которое состоит из клеток.
* Изображения для головы, хвоста и тела змейки, а также для фруктов.
* Отображение текущего счета и других игровых элементов.
* Взаимодействие с пользователем через кнопки, если они предусмотрены в интерфейсе.

**3. Работа с графикой и звуком**

Для создания графических элементов игры использованы изображения для головы и хвоста змейки, а также для фонов и фруктов. Звуковые эффекты, такие как звук при поедании фрукта, были добавлены для улучшения игрового опыта.

## **3.1. Разработка логики игры**

Так как наша игра ,где игрок управляет змейкой ,где змейка съедает пищу и проигрывает при столкновении с собой или со стеной. Мы используем Python, так как Pygame больше отвечает за графику .

**Движение змейки:**

def move\_snake(self):

        if self.new\_block == True:

            body\_copy = self.body[:]

            body\_copy.insert(0,body\_copy[0] + self.direction)

            self.body = body\_copy[:]

            self.new\_block = False

        else:

            body\_copy = self.body[:-1]

            body\_copy.insert(0,body\_copy[0] + self.direction)

            self.body = body\_copy[:]

**Съедание фрукта:**

  def check\_collision(self):

        if self.fruit.pos == self.snake.body[0]:

            self.fruit.randomize()

            self.snake.add\_block()

**Управление:**

if event.key == pygame.K\_w:

                if main\_game.snake.direction.y != 1:

                    main\_game.snake.direction = Vector2(0, -1)

**Проверка столкновений:**

def check\_fail(self):

        if not 0 <= self.snake.body[0].x < cell\_number or not 0 <= self.snake.body[0].y < cell\_number:

            self.game\_over()

        for block in self.snake.body[1:]:

            if block == self.snake.body[0]:

                self.game\_over()

**3.2.[Разработка графики и анимации](#_Toc190860011)**.

Для взаимодействия с Python и создание графики, звуков используется **Pygame**.

**1. Создаем графику и изображения.**

В игре используется тело змейки также фрукты и фон для игры. Все графические элементы загружаются с помощью библиотеки **Pygame** через метод pygame.image.load().

Примеры графических элементов:

* **Голова змейки.**
* **Тело змейки.**
* **Хвост змейки.**
* **Фрукты.**

**2.Анимация.**

**В игре это движущиеся объекты.При движении объектов обновляются только координаты объектов,а визуальное состояние остается статичным.**

def draw\_snake(self):

        for index,block in enumerate(self.body):

            x\_pos = int(block.x \* cell\_size)

            y\_pos = int(block.y \* cell\_size)

            block\_rect = pygame.Rect(x\_pos,y\_pos,cell\_size,cell\_size)

            if index == 0:

                screen.blit(self.head,block\_rect)

            elif index == len(self.body) - 1:

                screen.blit(self.tail,block\_rect)

**Звуковые эффекты.**

self.crunch\_sound = pygame.mixer.Sound('Sound/Sound\_crunch.wav')

def play\_crunch\_sound(self):

        self.crunch\_sound.play()

**Интерактивность и эффекты.**

Используется метод randomize(),которой изменяет координаты фрукта,при его съедании змейкой.

Def randomize(self):

        self.x = random.randint(0,cell\_number – 1)

        self.y = random.randint(0,cell\_number – 1)

        self.pos = Vector2(self.x,self.y)

**Графика и оформление.**

Графика которое изменяется под размер игрового поля. Это делает игру привлекательной и удобной для игры. Также фоновая графика такие как оформление различных кнопок и ячеек.

     self.background = pygame.image.load('Graphics/background.png').convert()

        self.background = pygame.transform.scale(self.background, (cell\_number \* cell\_size, cell\_number \* cell\_size))

## **3.3. Реализация игрового интерфейса.**

**Стартовый экран**.

В игре реализован стартовый экран, на котором отображается кнопка для начала игры. Стартовый экран отображается до тех пор, пока игрок не кликнет на кнопку для начала игры. Для этого используется класс **START\_SCREEN**, который загружает изображение фона и кнопки. При нажатии на кнопку запускается основная игровая логика.

class START\_SCREEN:

    def \_\_init\_\_(self):

        self.font = pygame.font.Font('Font/PoetsenOne-Regular.ttf', 40)

        self.button\_color = (0, 255, 0)

        self.button\_rect = pygame.Rect((cell\_number \* cell\_size // 4, cell\_number \* cell\_size // 2), (cell\_number \* cell\_size // 2, 50))

        self.background = pygame.image.load('Graphics/background.png').convert()

        self.background = pygame.transform.scale(self.background, (cell\_number \* cell\_size, cell\_number \* cell\_size))

        self.button\_background = pygame.image.load('Graphics/Button.png').convert\_alpha()

        self.button\_background = pygame.transform.scale(self.button\_background, (self.button\_rect.width, self.button\_rect.height))

    def draw(self):

        screen.blit(self.background, (0, 0))

        screen.blit(self.button\_background, self.button\_rect.topleft)

    def is\_button\_clicked(self, mouse\_pos):

        return self.button\_rect.collidepoint(mouse\_pos)

**Отображение очков**.

В игре реализовано отображение текущего счета, который отображается в нижней части экрана. Счет увеличивается с каждым съеденным фруктом. Для отображения используется шрифт, который обновляется каждый раз, когда змейка съедает фрукт.

 def draw\_score(self):

     score\_text = f"SCORE: {score}"

        score\_surface = game\_font.render(score\_text, True, (255, 255, 255))

        score\_rect = score\_surface.get\_rect(center=(cell\_number \* cell\_size // 2, cell\_number \* cell\_size // 2.5))

        screen.blit(score\_surface, score\_rect)

        screen.blit(self.restart\_button\_image, self.restart\_button.topleft)

**Обработчик кнопок и меню**.

На стартовом экране или в другом интерфейсе можно добавить кнопки для старта игры, паузы или выхода из игры. Эти кнопки можно обрабатывать с помощью проверки кликов мышью.

if event.type == pygame.MOUSEBUTTONDOWN and not game\_started:

            if event.button == 1:

                mouse\_pos = pygame.mouse.get\_pos()

                if start\_screen.is\_button\_clicked(mouse\_pos):

                    game\_started = True

**Перезапуск игры (после окончания)**:

После того как игрок проиграет (столкновение с собой или границами), можно реализовать экран с предложением перезапустить игру. Для этого можно использовать снова терминал.

**Интерфейс и управление игрой**.

Для управления игрой используется клавиатура, где **W, A, S, D** управляют движением змейки. Это простое управление, которое выполняется через обработку событий при нажатии клавиш.

if event.type == pygame.KEYDOWN and game\_started:

            if event.key == pygame.K\_w:

                if main\_game.snake.direction.y != 1:

                    main\_game.snake.direction = Vector2(0, -1)

            if event.key == pygame.K\_d:

                if main\_game.snake.direction.x != -1:

                    main\_game.snake.direction = Vector2(1, 0)

            if event.key == pygame.K\_s:

                if main\_game.snake.direction.y != -1:

                    main\_game.snake.direction = Vector2(0, 1)

            if event.key == pygame.K\_a:

                if main\_game.snake.direction.x != 1:

**Запуск проекта**

Чтобы запустить проект, на терминале выполняем:

Нажать на кнопку “Запуск” или “Run”

После этого приложение должно запуститься

# **4. Тестирование**

Тестирование системы включало несколько этапов:

1. **Юнит-тестирование** — тестирование отдельных методов игры (например, проверка логики движения змейки, появление фруктов или корректности работы с очками).
2. **Тестирование интерфейса** — проверка корректности отображения экрана ,интерфейсных элементов, такие как кнопки ,счет и другие компоненты игры.
3. **Нагрузочное тестирование** — проверка работы игры при большем количестве объектов(например, если змейка становится очень длинной) или тестирование производительности игры на различных устройствах.

Пример юнит-теста для функции движения позиции змейки:

import unittest

from pygame.math import Vector2

from snake\_game import SNAKE

class TestSnakeGame(unittest.TestCase):

def setUp(self):

self.snake = SNAKE()

def test\_move\_snake(self):

initial\_position = self.snake.body[0]

self.snake.direction = Vector2(1, 0)

self.snake.move\_snake()

new\_position = self.snake.body[0]

self.assertEqual(new\_position, initial\_position + Vector2(1, 0))

def test\_collision\_with\_fruit(self):

self.snake.body[0] = Vector2(5, 5) позицию фрукта

self.snake.direction = Vector2(0, 1)

self.snake.move\_snake()

self.assertEqual(len(self.snake.body), 4)

def test\_game\_over(self):

self.snake.body[0] = Vector2(0, 0)

self.snake.direction = Vector2(-1, 0)

self.snake.move\_snake()

self.assertTrue(self.snake.body[0].x < 0 or self.snake.body[0].y < 0)

Пример нагрузочного тестирования:

Может включать проверку производительности, например, когда змейка становится очень длинной, или когда на поле появляется много объектов. В этом случае проверяется, как система справляется с большими нагрузками.

# **5. Размещение в GitHub**

git init

git add .

git commit -m "Initial commit"

git remote add origin <репозиторий>

git push -u origin main

По завершению разработки система была размещена на GitHub. Это позволяет разработчикам легко отслеживать изменения, делиться кодом с коллегами и вносить улучшения. В репозитории добавлен файл .gitignore, чтобы исключить временные файлы и базы данных, и написаны четкие commit-сообщения для каждого значимого изменения.

# 

# **Заключение**

В ходе выполнения данной курсовой работы была разработана классическая игра Змейка, включающая в себя передвижение змейки, съедание пищи, увеличение длинны змейки, появление пищи кнопки и так далее. Разработка велась с использованием технологий Python и Pygame. Проект был протестирован и загружен в репозиторий GitHub для удобства.

**Сложности, возникшие в процессе разработки**

При разработке игры одной из наибольших трудностей стало проработать логику игрового процесса. Необходимо было создать не только правильную структуру данных для хранения состояния игры, но и максимально оптимизировать ее для быстрого обновления и обработки изменений в ней. На начальном этапе структура логики была довольно простой и все игровые элементы: змея, фрукты и блоки хранились в одном массиве. Однако после тестирования стало очевидно, что для управления состоянием игры нужно использовать более гибкое решение

Это касалось управления столкновениями змейки с телом и проверкой на столкновение с границей поля. Также возникли сложности с масштабированием игры: к примеру, когда змейка становилась больше, ее тело тоже увеличивалось, и необходимо было уметь обрабатывать тело змеи, не только голову. Это потребовало изменения структуры хранения данных путем добавления новых тел и оптимизации алгоритмов для обновления и взаимодействия объектов.

Аналогично, при разработке игры пришлось решать громадную проблему, связанную с созданием удобного и интуитивно понятного интерфейса. Поэтому придерживался точки зрения, что пользовательский опыт должен быть простым, однако при этом не потерять в функционале. Например, чтобы правильно описывать состояния самой игры: отображать счёт, состояние игрока, его рост, насыщение. Иметь возможность работать с клавиатурой. Для этого разработали различные элементы: кнопки, фон для отображения счёта и изображениях состояний игры.

Трудности также возникали с настройкой самого игрового процесса и взаимодействием между различными компонентами игры. Параметры интерфейса удавалось легко реализовать с помощью библиотеки Pygame, однако мне потребовалось время, чтобы освоить графику, анимацию и координацию отображения всех объектов на экране. Нестандартное и сложное для понимания поведение возникало при увеличении размера змейки, а также в процессе настройки столкновений и обработки их визуализации.В целом, все эти задачи требовали тщательной проработки логики игры и её интерфейса.

**Чему удалось научиться**

Во-первых, была приобретена практика работы с библиотекой Pygame. Этот инструмент оказался незаменимым для разработки 2D-игр, и теперь можно уверенно использовать его для создания графики, анимаций, а также обработки пользовательского ввода и событий в игре. Изучение Pygame позволило лучше понять принципы работы с игровыми объектами, а также улучшить навыки в создании интерактивных приложений.

Во-вторых, значительно улучшены навыки работы с логикой игры и оптимизацией процессов. Процесс создания структуры данных, обработки столкновений и управления состоянием игры позволил глубже понять основы алгоритмов и оптимизации для игровых приложений. Это также способствовало улучшению понимания работы с движениями объектов в 2D-пространстве и обработке сложных игровых сценариев.

Также была изучена работа с Git и GitHub. Это особенно важно при командной разработке, так как позволяет отслеживать изменения и возвращаться к предыдущим версиям проекта в случае необходимости.

**Итоги и достигнутые результаты**

В результате курсовой работы удалось создать полностью функционирующую игру "Змейка", которая включает в себя основные игровые механики и интерфейс. Основные задачи, поставленные перед проектом, были выполнены:

• Реализована логика игры: змейка управляется, правильно обрабатываются столкновения, и появляются новые блоки, когда она ест фрукты.  
• Создан удобный и понятный интерфейс с графикой, анимацией и отображение   
• Игра стабильно работает благодаря использованию библиотеки Pygame, что помогает эффективно управлять объектами и анимацией.  
• Добавлены звуковые эффекты и визуальные элементы для улучшения игры.  
• Игра тестировалась на разных этапах разработки, чтобы выявить ошибки, улучшить производительность и сделать игру более стабильной.  
• Проект завершен и готов к дальнейшему развитию, например, добавлению новых уровней или функций.

Эта работа не только позволила применить знания по разработке игр, но и дала полезный опыт в работе с графическими библиотеками и логикой игровых приложений. Создание игры "Змейка" стало отличной практикой, которая помогла освоить основы работы с Pygame и улучшить навыки программирования. В будущем этот опыт можно использовать для создания более сложных игр и добавления новых функций в уже существующие проекты.

# **Список литературы**

**1.** Митин, В. В. Разработка игр на Python с использованием Pygame / В. В. Митин. — М.: БХВ-Петербург, 2018. — 272 с.

**2.** Лутц, Марк. Изучаем Python / Марк Лутц. — 5-е изд. — М.: Вильямс, 2016. — 1600 с.

**3.** Мейерс, Дэвид. Python для начинающих. Пошаговое руководство по разработке игр / Дэвид Мейерс. — СПб.: Питер, 2019. — 352 с.

**4.** Шахид, Хасан. Game Development Using Python / Хасан Шахид. — Лондон: Packt Publishing, 2019. — 324 с.

# **Листинг**

import pygame, sys, random

from pygame.math import Vector2

pygame.mixer.pre\_init(44100, -16, 2, 512)

pygame.init()

cell\_size = 38

cell\_number = 19

screen = pygame.display.set\_mode((cell\_number \* cell\_size, cell\_number \* cell\_size))

clock = pygame.time.Clock()

grapes = pygame.image.load('Graphics/grapes.png').convert\_alpha()

game\_font = pygame.font.Font('Font/PoetsenOne-Regular.ttf', 25)

SCREEN\_UPDATE = pygame.USEREVENT

pygame.time.set\_timer(SCREEN\_UPDATE, 150)

class START\_SCREEN:

    def \_\_init\_\_(self):

        self.background = pygame.image.load('Graphics/background.png').convert()

        self.background = pygame.transform.scale(self.background, (cell\_number \* cell\_size, cell\_number \* cell\_size))

        self.start\_button = pygame.Rect((cell\_number \* cell\_size // 4, cell\_number \* cell\_size // 2), (cell\_number \* cell\_size // 2, 50))

        self.start\_button\_image = pygame.image.load('Graphics/Button.png').convert\_alpha()

        self.start\_button\_image = pygame.transform.scale(self.start\_button\_image, (self.start\_button.width, self.start\_button.height))

    def draw(self):

        screen.blit(self.background, (0, 0))

        screen.blit(self.start\_button\_image, self.start\_button.topleft)

    def is\_button\_clicked(self, mouse\_pos):

        if self.start\_button.collidepoint(mouse\_pos):

            return "start"

        return None

class SNAKE:

    def \_\_init\_\_(self):

        self.body = [Vector2(5, 10), Vector2(4, 10), Vector2(3, 10)]

        self.direction = Vector2(1, 0)

        self.new\_block = False

        self.head\_up = pygame.image.load('Graphics/head.up.png').convert\_alpha()

        self.head\_down = pygame.image.load('Graphics/head.down.png').convert\_alpha()

        self.head\_right = pygame.image.load('Graphics/head.right.png').convert\_alpha()

        self.head\_left = pygame.image.load('Graphics/head.left.png').convert\_alpha()

        self.tail\_up = pygame.image.load('Graphics/tail.up.png').convert\_alpha()

        self.tail\_down = pygame.image.load('Graphics/tail.down.png').convert\_alpha()

        self.tail\_right = pygame.image.load('Graphics/tail.right.png').convert\_alpha()

        self.tail\_left = pygame.image.load('Graphics/tail.left.png').convert\_alpha()

        self.body\_vertical = pygame.image.load('Graphics/body\_ver.png').convert\_alpha()

        self.body\_horizontal = pygame.image.load('Graphics/body\_hor1.png').convert\_alpha()

        self.body\_tr = pygame.image.load('Graphics/bl.dleft.png').convert\_alpha()

        self.body\_tl = pygame.image.load('Graphics/bl.dright.png').convert\_alpha()

        self.body\_br = pygame.image.load('Graphics/bl.left.png').convert\_alpha()

        self.body\_bl = pygame.image.load('Graphics/bl.right.png').convert\_alpha()

        self.crunch\_sound = pygame.mixer.Sound('Sound/eating.mp3')

    def draw\_snake(self):

        self.update\_head\_graphics()

        self.update\_tail\_graphics()

        for index, block in enumerate(self.body):

            x\_pos = int(block.x \* cell\_size)

            y\_pos = int(block.y \* cell\_size)

            block\_rect = pygame.Rect(x\_pos, y\_pos, cell\_size, cell\_size)

            if index == 0:

                screen.blit(self.head, block\_rect)

            elif index == len(self.body) - 1:

                screen.blit(self.tail, block\_rect)

            else:

                previous\_block = self.body[index + 1] - block

                next\_block = self.body[index - 1] - block

                if previous\_block.x == next\_block.x:

                    screen.blit(self.body\_vertical, block\_rect)

                elif previous\_block.y == next\_block.y:

                    screen.blit(self.body\_horizontal, block\_rect)

                else:

                    if previous\_block.x == -1 and next\_block.y == -1 or previous\_block.y == -1 and next\_block.x == -1:

                        screen.blit(self.body\_tl, block\_rect)

                    elif previous\_block.x == -1 and next\_block.y == 1 or previous\_block.y == 1 and next\_block.x == -1:

                        screen.blit(self.body\_bl, block\_rect)

                    elif previous\_block.x == 1 and next\_block.y == -1 or previous\_block.y == -1 and next\_block.x == 1:

                        screen.blit(self.body\_tr, block\_rect)

                    elif previous\_block.x == 1 and next\_block.y == 1 or previous\_block.y == 1 and next\_block.x == 1:

                        screen.blit(self.body\_br, block\_rect)

    def update\_head\_graphics(self):

        head\_relation = self.body[1] - self.body[0]

        if head\_relation == Vector2(1, 0):

            self.head = self.head\_left

        elif head\_relation == Vector2(-1, 0):

            self.head = self.head\_right

        elif head\_relation == Vector2(0, 1):

            self.head = self.head\_up

        elif head\_relation == Vector2(0, -1):

            self.head = self.head\_down

    def update\_tail\_graphics(self):

        tail\_relation = self.body[-2] - self.body[-1]

        if tail\_relation == Vector2(1, 0):

            self.tail = self.tail\_left

        elif tail\_relation == Vector2(-1, 0):

            self.tail = self.tail\_right

        elif tail\_relation == Vector2(0, 1):

            self.tail = self.tail\_up

        elif tail\_relation == Vector2(0, -1):

            self.tail = self.tail\_down

    def move\_snake(self):

        if self.new\_block:

            body\_copy = self.body[:]

            body\_copy.insert(0, body\_copy[0] + self.direction)

            self.body = body\_copy[:]

            self.new\_block = False

        else:

            body\_copy = self.body[:-1]

            body\_copy.insert(0, body\_copy[0] + self.direction)

            self.body = body\_copy[:]

    def add\_block(self):

        self.new\_block = True

    def play\_crunch\_sound(self):

        self.crunch\_sound.play()

    def reset(self):

        self.body = [Vector2(5, 10), Vector2(4, 10), Vector2(3, 10)]

        self.direction = Vector2(1, 0)

        self.new\_block=False

class FRUIT:

    def \_\_init\_\_(self):

        self.randomize()

    def draw\_fruit(self):

        fruit\_rect = pygame.Rect(int(self.pos.x \* cell\_size), int(self.pos.y \* cell\_size), cell\_size, cell\_size)

        screen.blit(grapes, fruit\_rect)

    def randomize(self):

        self.x = random.randint(0, cell\_number - 1)

        self.y = random.randint(0, cell\_number - 1)

        self.pos = Vector2(self.x, self.y)

class MAIN:

    def \_\_init\_\_(self):

        self.snake = SNAKE()

        self.fruit = FRUIT()

        self.game\_over = False

    def update(self):

        if not self.game\_over:

            self.snake.move\_snake()

            self.check\_collision()

            self.check\_fail()

    def draw\_elements(self):

        self.draw\_grass()

        self.fruit.draw\_fruit()

        self.snake.draw\_snake()

        self.draw\_score()

    def check\_collision(self):

        if self.fruit.pos == self.snake.body[0]:

            self.fruit.randomize()

            self.snake.add\_block()

            self.snake.play\_crunch\_sound()

        for block in self.snake.body[1:]:

            if block == self.fruit.pos:

                self.fruit.randomize()

    def check\_fail(self):

        if not 0 <= self.snake.body[0].x < cell\_number or not 0 <= self.snake.body[0].y < cell\_number:

            self.game\_over = True

            game\_over\_sound.play()

        for block in self.snake.body[1:]:

            if block == self.snake.body[0]:

                self.game\_over = True

                game\_over\_sound.play()

    def reset(self):

        self.snake.reset()

        self.fruit.randomize()

        self.game\_over = False

    def draw\_grass(self):

        grass\_color = (152, 255, 152)

        for row in range(cell\_number):

            if row % 2 == 0:

                for col in range(cell\_number):

                    if col % 2 == 0:

                        grass\_rect = pygame.Rect(col \* cell\_size, row \* cell\_size, cell\_size, cell\_size)

                        pygame.draw.rect(screen, grass\_color, grass\_rect)

            else:

                for col in range(cell\_number):

                    if col % 2 != 0:

                        grass\_rect = pygame.Rect(col \* cell\_size, row \* cell\_size, cell\_size, cell\_size)

                        pygame.draw.rect(screen, grass\_color, grass\_rect)

    def draw\_score(self):

        score\_text = str(len(self.snake.body) - 3)

        score\_surface = game\_font.render(score\_text, True, (56, 74, 12))

        score\_x = int(cell\_size \* cell\_number - 60)

        score\_y = int(cell\_size \* cell\_number - 40)

        score\_rect = score\_surface.get\_rect(center=(score\_x, score\_y))

        apple\_rect = grapes.get\_rect(midright=(score\_rect.left, score\_rect.centery))

        bg\_rect = pygame.Rect(apple\_rect.left, apple\_rect.top, apple\_rect.width + score\_rect.width + 6, apple\_rect.height)

        pygame.draw.rect(screen, (167, 209, 61), bg\_rect)

        screen.blit(score\_surface, score\_rect)

        screen.blit(grapes, apple\_rect)

        pygame.draw.rect(screen, (56, 74, 12), bg\_rect, 2)

class GAME\_OVER\_SCREEN:

    def \_\_init\_\_(self):

        self.background = pygame.image.load('Graphics/background2.png').convert()

        self.background = pygame.transform.scale(self.background, (cell\_number \* cell\_size, cell\_number \* cell\_size))

        self.restart\_button = pygame.Rect((cell\_number \* cell\_size // 4, cell\_number \* cell\_size // 2.2), (cell\_number \* cell\_size // 2, 50))

        self.restart\_button\_image = pygame.image.load('Graphics/Button.png').convert\_alpha()

        self.restart\_button\_image = pygame.transform.scale(self.restart\_button\_image, (self.restart\_button.width, self.restart\_button.height))

    def draw(self,score):

        screen.blit(self.background, (0, 0))

        game\_over\_text="GAME OVER"

        game\_over\_surface = game\_font.render(game\_over\_text, True, (255, 255, 255))

        game\_over\_rect = game\_over\_surface.get\_rect(center=(cell\_number \* cell\_size // 2, cell\_number \* cell\_size // 3))

        screen.blit(game\_over\_surface, game\_over\_rect)

        score\_text = f"SCORE: {score}"

        score\_surface = game\_font.render(score\_text, True, (255, 255, 255))

        score\_rect = score\_surface.get\_rect(center=(cell\_number \* cell\_size // 2, cell\_number \* cell\_size // 2.5))

        screen.blit(score\_surface, score\_rect)

        screen.blit(self.restart\_button\_image, self.restart\_button.topleft)

    def is\_button\_clicked(self, mouse\_pos):

        if self.restart\_button.collidepoint(mouse\_pos):

            return "restart"

        return None

game\_over\_sound=pygame.mixer.Sound('Sound/game.over.mp3')

start\_screen = START\_SCREEN()

game\_over\_screen = GAME\_OVER\_SCREEN()

main\_game = MAIN()

game\_started = False

in\_game\_over = False

while True:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            sys.exit()

        if event.type == SCREEN\_UPDATE and not in\_game\_over:

            main\_game.update()

        if event.type == pygame.MOUSEBUTTONDOWN:

            if not game\_started:

                mouse\_pos = pygame.mouse.get\_pos()

                button\_clicked = start\_screen.is\_button\_clicked(mouse\_pos)

                if button\_clicked == "start":

                    game\_started = True

            elif in\_game\_over:

                mouse\_pos = pygame.mouse.get\_pos()

                button\_clicked = game\_over\_screen.is\_button\_clicked(mouse\_pos)

                if button\_clicked == "restart":

                    main\_game.reset()

                    in\_game\_over = False

                    game\_started = True

        if event.type == pygame.KEYDOWN and not in\_game\_over:

            if event.key == pygame.K\_w:

                if main\_game.snake.direction.y != 1:

                    main\_game.snake.direction = Vector2(0, -1)

            if event.key == pygame.K\_d:

                if main\_game.snake.direction.x != -1:

                    main\_game.snake.direction = Vector2(1, 0)

            if event.key == pygame.K\_s:

                if main\_game.snake.direction.y != -1:

                    main\_game.snake.direction = Vector2(0, 1)

            if event.key == pygame.K\_a:

                if main\_game.snake.direction.x != 1:

                    main\_game.snake.direction = Vector2(-1, 0)

    if not game\_started:

        start\_screen.draw()

    elif main\_game.game\_over:

        in\_game\_over = True

        game\_over\_screen.draw(len(main\_game.snake.body)-3)

    else:

        screen.fill((85, 107, 47))

        main\_game.draw\_elements()

    pygame.display.update()

    clock.tick(60)