### **Лабораторная работа №3: Сложная анимация**

[АиСД и ПП (2025-26, осень)](https://colab.research.google.com/drive/1ntz7NYJu3oa4WkR9RWlJioqjohyLzkpd#scrollTo=DWTJJ9NK4bAq)

### **Срок сдачи:** [25/11/2025]

#### **1. Цель работы**

Основная цель этой лабораторной работы применить ваши навыки работы с графикой и анимацией к более сложным проектам. Каждая задача требует тщательного **проектирования и планирования программы** перед реализацией. Вам потребуется исследовать математические концепции и создавать хорошо структурированный, профессиональный код.

#### **2. Порядок выполнения**

* **Индивидуальная работа:** Это **индивидуальное задание**. Задачи будут случайным образом распределены преподавателем.
* **Консультации:** На семинарах вы можете попросить преподавателя пояснить требования или математические концепции.
* **Творческий подход:** Каждая задача предоставляет только общую основу. Вы должны принимать дизайнерские решения относительно:
  + Скорости и плавности анимации
  + Размеров элементов и цветовых схем
  + Визуального стиля и пользовательского опыта
  + Преподаватель будет оценивать ваши дизайнерские решения — удачные решения повысят оценку, неудачные, снизят.
* **Исследовательский компонент:** Многие задачи требуют понимания математических концепций. Исследование, необходимая часть успешного выполнения этих проектов.

#### **3. Требования к качеству кода**

* **Требуется ООП:** Вы должны использовать принципы объектно-ориентированного программирования для организации кода.
* **Качество кода важно:** Ваш код должен соответствовать стилевым соглашениям, содержать комментарии и быть читаемым.
* **Рекомендуется система контроля версий:** Использование Git или подобных систем приветствуется и будет отмечено.
* **Управление настройками:** Используйте конфигурационные файлы (JSON/YAML) для хранения настроек, таких как размер экрана, цвета и скорости.

**Пример config.json:**

json

{

"window\_width": 800,

"window\_height": 600,

"animation\_speed": 60,

"background\_color": [0, 0, 0],

"gravity\_strength": 9.8

}

#### **4. Дополнительные задания (по желанию)**

Каждая задача включает пронумерованные дополнительные требования. Вы также можете предложить собственные расширения. Дополнительная функциональность будет оцениваться на основе сложности и качества реализации.

Дополнительные задания часто включают:

* Обработку событий
* Элементы пользовательского интерфейса
* Взаимодействие с мышью и клавиатурой
* Ввод/вывод файлов

Эти навыки важны для будущих заданий.

#### **5. Требования к отчету (PDF-файл)**

Покажите мне один аккуратно оформленный PDF-файл до дедлайна.

**A. Титульная страница**

* ФИО студента
* Номер студенческого билета
* Название предмета и код
* Номер лабораторной работы (Лаба 3)
* Номер группы
* Дата сдачи
* Название назначенной задачи

**B. Описание задания**

* Напишите полный текст вашей задачи.

**C. Достигнутая сложность и реализация**

* Перечислите выполненные основные требования
* Перечислите выполненную дополнительную функциональность
* Укажите любые оригинальные расширения, которые вы добавили

**D. Объяснение проектирования программы**

* **Структура ООП:** Опишите созданные классы и их обязанности
* **Организация кода:** Объясните, как вы структурировали код
* **Математическое исследование:** Объясните исследованные и реализованные математические концепции
* **Управление настройками:** Покажите ваш конфигурационный файл и объясните его использование

**E. Основная часть (структурированная по требованиям)**Для каждого требования (основного и дополнительного):

1. **Требование:** Что нужно было реализовать
2. **Решение:** Как вы это реализовали, включая возникшие трудности
3. **Фрагмент кода:** Соответствующий, закомментированный код
4. **Скриншот:** Визуальное доказательство работоспособности

**F. Полный исходный код**

* Включите ваш полный, окончательный код на Python

#### **6. Критерии оценки и защита работы**

Ваша способность понимать и изменять ваш код во время защиты существенно влияет на итоговую оценку.

**Академическая честность:** Вся работа должна быть вашей собственной.

**Начните с исследования и планирования. Спроектируйте ваши классы до написания кода. Тщательно тестируйте и будьте готовы объяснить каждую часть вашей реализации**

#### **6. Список заданий**

Вам будет случайным образом назначена одна из этих задач:

### **Сложная анимация (22)**

| *1- Фейерверки (152)* На экране случайно генерируются фейерверки. Сам фейерверк и его частицы имеют ограниченное время жизни.   1. Добавьте генерацию фейерверков разных цветов и прозрачность. 2. Поэкспериментируйте с гравитацией частиц. |  |
| --- | --- |
| *2- Генератор лабиринтов (166)* Реализуйте [алгоритм генерации лабиринтов](https://habr.com/ru/post/262345/) через поиск в ширину.   1. Реализуйте разные размеры лабиринтов. 2. Реализуйте экспорт получившихся лабиринтов во внешний файл. |  |
| *3- Алгоритм A\* (254)* Реализуйте алгоритм поиска кратчайшего пути на сетке. Обращайте разными цветами посещённые и текущие клетки сетки.   1. Реализуйте импорт лабиринтов из внешнего файла. 2. Реализуйте другие популярные алгоритмы поиска пути. |  |
| *4- Притяжение (113)* Реализуйте систему частиц, каждая из которых притягивается к определенному месту на экране. Одновременно, каждая частица отталкивания от текущих координат мышки. Должно получиться, что курсор "разбрасывает" частицы от себя.   1. Нарисуйте местами притяжения частиц какой-нибудь рисунок или надпись. 2. Реализуйте импорт рисунка из внешнего файла. |  |
| *5- Превращение круга (59)* С помощью полярных координат реализуйте следующее преобразование круга в треугольник. |  |
| *Анимация круга (78)* С помощью полярных координат реализующее превращение круга в треугольник как на приведенной анимации. |  |
| *6- Игра жизнь (89)* Реализуйте известный клеточный автомат под названием игра "жизнь". Используйте тороидальную геометрию. Начните м небольшого количества клеток.   1. Реализуйте на том же движке игру по альтернативными правилам. 2. Реализуйте возможность добавления известных фигур на поле перетаскиванием. |  |
| *7-* [*Папоротник Барнсли*](https://en.wikipedia.org/wiki/Barnsley_fern#Construction) *(48)* При помощи интерактивного алгоритма постройте из точек фрактальную фигуру.   1. Манипулируя параметрами уравнения, получите другие виды фрактальных растений. |  |
| *8-* [*Последовательность Ракамана*](https://en.wikipedia.org/wiki/Recam%C3%A1n%27s_sequence) *(89)* Реализуйте популярную визуализацию последовательности Ракамана через полуокружности на числовой прямой.  [Видео](https://www.youtube.com/watch?v=FGC5TdIiT9U) в помощь.   1. Таким же образом визуализируйте другие известные натуральные последовательности. 2. Реализуйте автоматическое масштабирование как на приведенной анимации. |  |
| *9-* [*Бойды*](https://en.wikipedia.org/wiki/Boids) *(165)* Реализуйте известный алгоритм "бойды", стимулирующий стайное проведение кому и других животных. Используйте набор бойдов со случайными положением и случайной, но органичной начальной скоростью.   1. Добавьте возможность включить или отключать каждую из трёх входящих в алгоритм сил, например, по нажатию кнопки клавиатуры 2. Позвольте пользователю регулировать весе этих трёх сил, например при помощи слайдеров. |  |
| *10- Ряд Фурье (58)* Реализуйте визуализацию рядов Фурье для [прямоугольной](https://www.mathsisfun.com/calculus/fourier-series.html) функции. [Видео](https://www.youtube.com/watch?v=ds0cmAV-Yek) в поддержку. Начните с небольшого количества членов ряда.   1. Позвольте пользователю регулировать количество членов ряда и, как следствие, точность воспроизведения функции. 2. Реализуйте отрисовку таким же образом других функций, например, пилообразной. |  |
| *11-* [*Фрактал Омара Пола*](https://en.wikipedia.org/wiki/Toothpick_sequence) *(149)* Нарисуйте при помощи рекурсивного или циклического алгоритма известный фрактальный узор Омара Пола, также известный как рисунок из зубочисток.  [Видео](https://www.youtube.com/watch?v=_UtCli1SgjI) в поддержку.   1. Реализуйте автоматический зум, как на приведенной анимации. |  |
| *12-* [*Кривая Коха*](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%B2%D0%B0%D1%8F_%D0%9A%D0%BE%D1%85%D0%B0) *(93)* Реализуйте при помощи рекурсивного алгоритма отрисовку снежинки Коха. Отрегулируйте глубину рекурсии в зависимости от размера получаемых на текущей итерации элементов.   1. Реализуйте похожим образом другие известные фрактальные кривые, например, кривую дракона. 2. Отрефакторите код таким образом, чтобы алгоритм построения фрактала и логика отрисовки были изолированными частями программы. |  |
| *13 - Черная дыра (155)* Реализуйте отрисовку стимуляции параллельного пучка фотонов, подходящего вблизи черной дыры и искривляющегося под действием ее гравитации.   1. Позвольте пользователю регулировать массу черной дыры при помощи слайдера. |  |
| *14 -* [*Кривая Гильберта*](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%B2%D0%B0%D1%8F_%D0%93%D0%B8%D0%BB%D1%8C%D0%B1%D0%B5%D1%80%D1%82%D0%B0) *(90)* Реализуйте отрисовку заполнения пространства кривой Гильберта. Используйте рекурсивный алгоритм. Начните с малого количества итераций.   1. Добавьте плавное изменение цвета линии как на приведенной анимации. 2. Таким же образом визуализируйте другие заполняющие плоскость кривые. |  |
| *15 - Гравитация (99)* Используя законы Ньютона реализуйте систему спутников, вращающихся вокруг центрального тела. Спутники брать безмассовыми.   1. Добавьте возможность упругого столкновения спутников. 2. Реализуйте этот алгоритм в тех измерениях. |  |
| *16 -( )Силы* [*сопротивления*](https://ru.wikipedia.org/wiki/%D0%9B%D0%BE%D0%B1%D0%BE%D0%B2%D0%BE%D0%B5_%D1%81%D0%BE%D0%BF%D1%80%D0%BE%D1%82%D0%B8%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) *(102)* Реализуйте систему разноразмерных частиц, имеющих одинаковую вертикальную скорость. При прохождении границы сред частицами должны испытывать сопротивление.   1. Реализуйте более физически достоверную модель сопротивления, зависящего от размера частицы. 2. Добавьте учёт инерции частиц в зависимости от их массы. |  |
| *17 - Упругое столкновение (80)* Реализуйте механизм упругого столкновения круглой частицы с плоской границей экрана, как показано на приведенной анимации. |  |

### —---------------------------------------------------------

**7 - Как подходить к любой задаче:**

**Шаг 1: Понять суть задания**

Выделите ключевые слова (например: "фейерверки", "частицы", "время жизни")

Найдите дополнительные требования ("разные цвета", "прозрачность", "гравитация")

**Шаг 2: Исследовать математическую основу**

Поищите в Google: "[название задачи] математика простое объяснение"

Посмотрите видео на YouTube об алгоритме

Найдите визуальные примеры того, что нужно создать

**Шаг 3: Разработать ООП структуру (САМОЕ ВАЖНОЕ)**

# Шаблон для ЛЮБОЙ задачи

class Simulation:

def update(self): pass # Здесь происходит физика/математика

def draw(self): pass # Здесь создается визуальная красота

class Manager:

def \_\_init\_\_(self): pass # Управляет всем

**Шаг 4: Создать config.json**

json

{

"window\_size": [800, 600],

"colors": {"main": [255,0,0], "background": [0,0,0]},

"speeds": {"animation": 60, "particles": 5.0},

"behavior": {"gravity": 0.5, "lifetime": 2.0}

}

**Дополнительные советы:**

**Для успешного выполнения:**

Начните с простого - сначала сделайте базовую работающую версию

Тестируйте каждый шаг - не пишите весь код сразу

Используйте комментарии - объясняйте, что делает каждая часть кода

Создавайте красивые визуальные эффекты - это повысит оценку

**Профессиональные приемы:**

Храните все числа в config.json - размеры, скорости, цвета

Добавьте управление с клавиатуры - пробел для паузы, R для сброса

Сделайте плавную анимацию - стремитесь к 60 кадрам в секунду

Для отчета:

Делайте скриншоты на каждом этапе

Сохраняйте разные версии кода

Записывайте возникшие проблемы и их решения

Объясняйте свои дизайнерские решения

Этот подход поможет вам систематически работать над любой из 17 задач и создать качественную лабораторную работу!