МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ижевский государственный технический университет имени М.Т. Калашникова»

(ФГБОУ ВО «ИжГТУ имени М.Т. Калашникова»)

Факультет «Информатика и вычислительная техника»

Кафедра «Программное обеспечение»

Направление подготовки 09.03.04 «Программная инженерия»

ОТЧЕТ ПО ПРАКТИКЕ

|  |
| --- |
|  |
| *(наименование типа практики)* |
| *(полное наименование профильной организации)*  Выполнил обучающийся /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Нуруллин Эльмир Рафисович, 1 курс, Б23-191-2 |
| *(подпись) (ФИО, курс, номер группы)*  Дата сдачи отчета: « » 2024 г. |
| Дата аттестации « » 20 г. |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Руководитель практики от  ИЖГТУ имени М.Т. Калашникова / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ |
| *(подпись) (И.О. Фамилия, должность, ученая степень)*  Заведующий кафедрой /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ |
| *(подпись) (И.О. Фамилия, должность, ученая степень)* |

СОДЕРАЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc168698486)

[1. ОСНОВНАЯ ЧАСТЬ 4](#_Toc168698487)

[1.1 Словесное описание проекта, постановка задачи 4](#_Toc168698488)

[1.2 Математическая постановка 4](#_Toc168698489)

[1.3 Описание программной реализации 5](#_Toc168698490)

[1.4 Листинг текста программы, тестовые примеры работы 9](#_Toc168698491)

[ЗАКЛЮЧЕНИЕ 16](#_Toc168698492)

[2. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 17](#_Toc168698493)

ВВЕДЕНИЕ

Цель проекта - разработка консольного приложения, позволяющего

работать с графическими объектами и добавлять анимационные эффекты на произвольную тему. В процессе выполнения проекта будут изучены теоретические основы создания анимации, а также реализована анимация на выбранную тему.

Цель практики - закрепление пройденного материала, а также самостоятельное изучение дополнительной информации.

Задачи учебной практики:

1. Реализация анимационных эффектов.
2. Реализация обработки событий клавиатуры (управление клавишами).
3. Использование рекурсии (фрактала).
4. Использование случайных чисел.

1. ОСНОВНАЯ ЧАСТЬ

1.1 Словесное описание проекта

В данной работе рассматривается создание простой, но увлекательной игры на основе механики, подобной популярной игре "Flappy Bird". Основная цель проекта — разработка игры с использованием библиотеки Pygame, включающей в себя элементы фрактального искусства.

Постановка задачи

Проект представляет собой аркадную игру, где игрок управляет самолетом, который должен избегать столкновений с препятствиями, не касаться земли и неба. Игровое поле также украшено фракталом — снежинкой Коха (после нажатия кнопки “R” на клавиатуре, когда игра не активна), которая изменяет свою форму в зависимости от времени. Цель игры — продержаться как можно дольше, набирая очки за каждую секунду, проведенную в игре.

Основные задачи:

1. Разработка игровой механики управления самолетом.
2. Создание динамических препятствий.
3. Включение фрактального элемента на игровом поле.
4. Обеспечение звукового сопровождения и интерфейса пользователя.
5. Оптимизация производительности игры.

1.2 Математическая постановка

В основе игры лежит несколько математических задач:

**Движение самолета под воздействием гравитации**:

Позиция самолета обновляется по формуле: , где ​ — начальная позиция, — скорость, g — гравитация.

При прыжке скорость изменяется: ​.

**Создание и обновление фрактала (снежинка Коха)**:

Алгоритм рекурсивного построения фрактала: каждая линия делится на три части, средняя треть заменяется равносторонним треугольником, стоящим на этой линии.

**Перемещение фона и препятствий**:

Скорость движения объектов задана константами и изменяется с течением времени:.

1.3 Описание программной реализации

Программная реализация игры включает несколько классов:

1. **Структура данных (классы):**

1. **BG** — класс для заднего фона.
2. **Ground** — класс для земли.
3. **Plane** — класс для управления самолетом.
4. **Obstacle** — класс для динамических препятствий.

Инициализация игры происходит с использованием библиотеки Pygame. Игровой цикл включает обработку событий, обновление состояния объектов и отрисовку на экране.

Класс **Game** содержит основной цикл игры и методы для обработки коллизий, отображения счета и рисования фрактала. Фрактал рисуется с использованием рекурсивного алгоритма, где каждое новое положение линий рассчитывается на основе текущего угла.

2. **Функции**:

**koch\_snowflake(start\_pos, end\_pos, depth, angle\_offset)**

* Рисует фрактал "снежинка Коха" рекурсивным методом.

**Game.init(self)**

* Инициализирует основной игровой класс, создает все необходимые объекты и настраивает начальные параметры.
* Создает игровые группы спрайтов, загружает фоновую музыку и меню, устанавливает начальные значения для счета и других игровых параметров.

**Game.collisions(self)**

* Обрабатывает столкновения самолета с препятствиями или верхней границей экрана.
* Если происходит столкновение, останавливает игру и удаляет все препятствия.

**Game.display\_score(self)**

* Отображает текущий счет игрока на экране.
* Счет отображается в разных местах экрана в зависимости от того, активна ли игра или нет.

**Game.draw\_koch\_snowflake(self)**

* Рисует фрактал "снежинка Коха" на экране.
* Угол поворота фрактала увеличивается с каждым кадром, создавая анимацию вращения.

**Game.run(self)**

* Основной игровой цикл.
* Обрабатывает события, обновляет состояния объектов, отрисовывает их на экране и контролирует частоту кадров.
* Если игра активна, проверяет столкновения и обновляет счет.

**BG.init(self, groups, scale\_factor)**

* Инициализирует спрайт фона.
* Загружает изображение фона, масштабирует его и создает дублирующее изображение для эффекта бесконечной прокрутки.
* Устанавливает начальную позицию.

**BG.update(self, dt)**

* Обновляет положение фона, создавая эффект его движения влево.
* Если фон полностью смещается влево, сбрасывает его позицию для бесконечной прокрутки.

**Ground.init(self, groups, scale\_factor)**

* Инициализирует спрайт земли.
* Загружает и масштабирует изображение земли, устанавливает начальную позицию и создает маску для коллизий.

**Ground.update(self, dt)**

* Обновляет положение земли, перемещая ее влево.
* Если земля полностью смещается влево, сбрасывает ее позицию для бесконечной прокрутки.

**Plane.init(self, groups, scale\_factor)**

* Инициализирует спрайт самолета.
* Загружает кадры анимации самолета, устанавливает начальную позицию и параметры движения (гравитация, направление).
* Загружает звук прыжка и создает маску для коллизий.

**Plane.import\_frames(self, scale\_factor)**

* Загружает и масштабирует кадры анимации самолета.

**Plane.apply\_gravity(self, dt)**

* Применяет эффект гравитации к самолету, изменяя его вертикальное положение в зависимости от времени.

**Plane.jump(self)**

* Обрабатывает прыжок самолета при нажатии кнопки.
* Проигрывает звук прыжка и изменяет направление движения самолета.

**Plane.animate(self, dt)**

* Обновляет текущий кадр анимации самолета.

**Plane.rotate(self)**

* Вращает изображение самолета в зависимости от направления его движения.
* Обновляет маску для коллизий после вращения.

**Plane.update(self, dt)**

* Обновляет состояние самолета, применяя гравитацию, анимацию и вращение.

**Obstacle.init(self, groups, scale\_factor)**

* Инициализирует спрайт препятствия.
* Выбирает случайную ориентацию (вверх или вниз), загружает и масштабирует изображение.
* Устанавливает начальную позицию и создает маску для коллизий.

**Obstacle.update(self, dt)**

* Обновляет положение препятствия, перемещая его влево.
* Если препятствие выходит за пределы экрана, оно удаляется.

**3. Основной цикл программы:**

1) Обработка событий, таких как нажатие клавиш и кнопок мыши.

2) Обновление и отрисовка объектов на экране.

3) Проверка столкновений и завершение игры при необходимости.

4) Обновление и отображение счета игрока.

5) Отображение фрактала снежинки Коха при завершении игры и нажатии определенной клавиши.

**4. Визуализация:**  
 • Использование текстур для отображения скал, самолёта и заднего фона.  
 • Позиционирование каждого элемента на экране в соответствии с его координатами на игровом поле.

**5. Управление:**

• для маневрирования между скалами используются одиночные или многократные нажатия левой кнопки мыши для прыжка

• после проигрыша при нажатии клавиши “R” в верхней части появиться фрактал “Снежинка Коха”.

1.4 Листинг текста программы

Файл main.py:

import pygame  
from settings import \*  
from random import choice, randint  
  
*# Класс для заднего фона*class BG(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, groups, scale\_factor):  
 super().\_\_init\_\_(groups)  
 bg\_image = pygame.image.load('graphics/environment/background (1).png').convert()  
  
 *# Масштабирование изображения фона до нужного размера* full\_height = bg\_image.get\_height() \* scale\_factor  
 full\_width = bg\_image.get\_width() \* scale\_factor  
 full\_sized\_image = pygame.transform.scale(bg\_image, (full\_width, full\_height))  
  
 *# Создание поверхности для изображения фона и дублирование для эффекта прокрутки* self.image = pygame.Surface((full\_width \* 2, full\_height))  
 self.image.blit(full\_sized\_image, (0, 0))  
 self.image.blit(full\_sized\_image, (full\_width, 0))  
  
 *# Установка прямоугольника и позиции* self.rect = self.image.get\_rect(topleft=(0, 0))  
 self.pos = pygame.math.Vector2(self.rect.topleft)  
  
 def update(self, dt):  
 *# Перемещение фона влево* self.pos.x -= 300 \* dt  
 if self.rect.centerx <= 0:  
 self.pos.x = 0 *# Сброс позиции при полном смещении* self.rect.x = round(self.pos.x)  
  
*# Класс для земли*class Ground(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, groups, scale\_factor):  
 super().\_\_init\_\_(groups)  
 self.sprite\_type = 'ground'  
  
 *# Загрузка и масштабирование изображения земли* ground\_surf = pygame.image.load('graphics/environment/ground (1).png').convert\_alpha()  
 self.image = pygame.transform.scale(ground\_surf, pygame.math.Vector2(ground\_surf.get\_size()) \* scale\_factor)  
  
 *# Установка позиции земли* self.rect = self.image.get\_rect(bottomleft=(0, WINDOW\_HEIGHT))  
 self.pos = pygame.math.Vector2(self.rect.topleft)  
  
 *# Создание маски для коллизий* self.mask = pygame.mask.from\_surface(self.image)  
  
 def update(self, dt):  
 *# Перемещение земли влево* self.pos.x -= 360 \* dt  
 if self.rect.centerx <= 0:  
 self.pos.x = 0 *# Сброс позиции при полном смещении* self.rect.x = round(self.pos.x)  
  
*# Класс для самолета*class Plane(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, groups, scale\_factor):  
 super().\_\_init\_\_(groups)  
  
 *# Загрузка кадров анимации самолета* self.import\_frames(scale\_factor)  
 self.frame\_index = 0  
 self.image = self.frames[self.frame\_index]  
  
 *# Установка прямоугольника и позиции* self.rect = self.image.get\_rect(midleft=(WINDOW\_WIDTH / 20, WINDOW\_HEIGHT / 2))  
 self.pos = pygame.math.Vector2(self.rect.topleft)  
  
 *# Параметры движения* self.gravity = 600  
 self.direction = 0  
  
 *# Создание маски для коллизий* self.mask = pygame.mask.from\_surface(self.image)  
  
 *# Загрузка звука прыжка* self.jump\_sound = pygame.mixer.Sound('sounds/jump.wav')  
 self.jump\_sound.set\_volume(0.3)  
  
 def import\_frames(self, scale\_factor):  
 *# Импорт кадров анимации самолета* self.frames = []  
 for i in range(3):  
 surf = pygame.image.load(f'graphics/plane/red{i}.png').convert\_alpha()  
 scaled\_surface = pygame.transform.scale(surf, pygame.math.Vector2(surf.get\_size()) \* scale\_factor)  
 self.frames.append(scaled\_surface)  
  
 def apply\_gravity(self, dt):  
 *# Применение гравитации к самолету* self.direction += self.gravity \* dt  
 self.pos.y += self.direction \* dt  
 self.rect.y = round(self.pos.y)  
  
 def jump(self):  
 *# Прыжок самолета при нажатии кнопки* self.jump\_sound.play()  
 self.direction = -400  
  
 def animate(self, dt):  
 *# Анимация самолета* self.frame\_index += 10 \* dt  
 if self.frame\_index >= len(self.frames):  
 self.frame\_index = 0  
 self.image = self.frames[int(self.frame\_index)]  
  
 def rotate(self):  
 *# Вращение самолета в зависимости от направления движения* rotated\_plane = pygame.transform.rotozoom(self.image, -self.direction \* 0.06, 1)  
 self.image = rotated\_plane  
 self.mask = pygame.mask.from\_surface(self.image)  
  
 def update(self, dt):  
 *# Обновление состояния самолета* self.apply\_gravity(dt)  
 self.animate(dt)  
 self.rotate()  
  
*# Класс для препятствий*class Obstacle(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, groups, scale\_factor):  
 super().\_\_init\_\_(groups)  
 self.sprite\_type = 'obstacle'  
  
 *# Выбор ориентации препятствия (вверх или вниз)* orientation = choice(('up', 'down'))  
 surf = pygame.image.load(f'graphics/obstacles/{choice((0, 1))}.png').convert\_alpha()  
 self.image = pygame.transform.scale(surf, pygame.math.Vector2(surf.get\_size()) \* scale\_factor)  
  
 *# Установка позиции препятствия за пределами экрана* x = WINDOW\_WIDTH + randint(40, 100)  
  
 if orientation == 'up':  
 *# Препятствие снизу* y = WINDOW\_HEIGHT + randint(10, 50)  
 self.rect = self.image.get\_rect(midbottom=(x, y))  
 else:  
 *# Препятствие сверху* y = randint(-50, -10)  
 self.image = pygame.transform.flip(self.image, False, True)  
 self.rect = self.image.get\_rect(midtop=(x, y))  
  
 self.pos = pygame.math.Vector2(self.rect.topleft)  
  
 *# Создание маски для коллизий* self.mask = pygame.mask.from\_surface(self.image)  
  
 def update(self, dt):  
 *# Перемещение препятствия влево* self.pos.x -= 400 \* dt  
 self.rect.x = round(self.pos.x)  
 if self.rect.right <= -100:  
 self.kill() *# Удаление препятствия, когда оно выходит за пределы экрана*

Файл settings.py:

WINDOW\_WIDTH = 480  
WINDOW\_HEIGHT = 800  
FRAMERATE = 120 *#частота кадрова*

Файл sprities.py:

import pygame  
from settings import \*  
from random import choice, randint  
  
*# Класс для заднего фона*class BG(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, groups, scale\_factor):  
 super().\_\_init\_\_(groups)  
 bg\_image = pygame.image.load('graphics/environment/background (1).png').convert()  
  
 *# Масштабирование изображения фона до нужного размера* full\_height = bg\_image.get\_height() \* scale\_factor  
 full\_width = bg\_image.get\_width() \* scale\_factor  
 full\_sized\_image = pygame.transform.scale(bg\_image, (full\_width, full\_height))  
  
 *# Создание поверхности для изображения фона и дублирование для эффекта прокрутки* self.image = pygame.Surface((full\_width \* 2, full\_height))  
 self.image.blit(full\_sized\_image, (0, 0))  
 self.image.blit(full\_sized\_image, (full\_width, 0))  
  
 *# Установка прямоугольника и позиции* self.rect = self.image.get\_rect(topleft=(0, 0))  
 self.pos = pygame.math.Vector2(self.rect.topleft)  
  
 def update(self, dt):  
 *# Перемещение фона влево* self.pos.x -= 300 \* dt  
 if self.rect.centerx <= 0:  
 self.pos.x = 0 *# Сброс позиции при полном смещении* self.rect.x = round(self.pos.x)  
  
*# Класс для земли*class Ground(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, groups, scale\_factor):  
 super().\_\_init\_\_(groups)  
 self.sprite\_type = 'ground'  
  
 *# Загрузка и масштабирование изображения земли* ground\_surf = pygame.image.load('graphics/environment/ground (1).png').convert\_alpha()  
 self.image = pygame.transform.scale(ground\_surf, pygame.math.Vector2(ground\_surf.get\_size()) \* scale\_factor)  
  
 *# Установка позиции земли* self.rect = self.image.get\_rect(bottomleft=(0, WINDOW\_HEIGHT))  
 self.pos = pygame.math.Vector2(self.rect.topleft)  
  
 *# Создание маски для коллизий* self.mask = pygame.mask.from\_surface(self.image)  
  
 def update(self, dt):  
 *# Перемещение земли влево* self.pos.x -= 360 \* dt  
 if self.rect.centerx <= 0:  
 self.pos.x = 0 *# Сброс позиции при полном смещении* self.rect.x = round(self.pos.x)  
  
*# Класс для самолета*class Plane(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, groups, scale\_factor):  
 super().\_\_init\_\_(groups)  
  
 *# Загрузка кадров анимации самолета* self.import\_frames(scale\_factor)  
 self.frame\_index = 0  
 self.image = self.frames[self.frame\_index]  
  
 *# Установка прямоугольника и позиции* self.rect = self.image.get\_rect(midleft=(WINDOW\_WIDTH / 20, WINDOW\_HEIGHT / 2))  
 self.pos = pygame.math.Vector2(self.rect.topleft)  
  
 *# Параметры движения* self.gravity = 600  
 self.direction = 0  
  
 *# Создание маски для коллизий* self.mask = pygame.mask.from\_surface(self.image)  
  
 *# Загрузка звука прыжка* self.jump\_sound = pygame.mixer.Sound('sounds/jump.wav')  
 self.jump\_sound.set\_volume(0.3)  
  
 def import\_frames(self, scale\_factor):  
 *# Импорт кадров анимации самолета* self.frames = []  
 for i in range(3):  
 surf = pygame.image.load(f'graphics/plane/red{i}.png').convert\_alpha()  
 scaled\_surface = pygame.transform.scale(surf, pygame.math.Vector2(surf.get\_size()) \* scale\_factor)  
 self.frames.append(scaled\_surface)  
  
 def apply\_gravity(self, dt):  
 *# Применение гравитации к самолету* self.direction += self.gravity \* dt  
 self.pos.y += self.direction \* dt  
 self.rect.y = round(self.pos.y)  
  
 def jump(self):  
 *# Прыжок самолета при нажатии кнопки* self.jump\_sound.play()  
 self.direction = -400  
  
 def animate(self, dt):  
 *# Анимация самолета* self.frame\_index += 10 \* dt  
 if self.frame\_index >= len(self.frames):  
 self.frame\_index = 0  
 self.image = self.frames[int(self.frame\_index)]  
  
 def rotate(self):  
 *# Вращение самолета в зависимости от направления движения* rotated\_plane = pygame.transform.rotozoom(self.image, -self.direction \* 0.06, 1)  
 self.image = rotated\_plane  
 self.mask = pygame.mask.from\_surface(self.image)  
  
 def update(self, dt):  
 *# Обновление состояния самолета* self.apply\_gravity(dt)  
 self.animate(dt)  
 self.rotate()  
  
*# Класс для препятствий*class Obstacle(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, groups, scale\_factor):  
 super().\_\_init\_\_(groups)  
 self.sprite\_type = 'obstacle'  
  
 *# Выбор ориентации препятствия (вверх или вниз)* orientation = choice(('up', 'down'))  
 surf = pygame.image.load(f'graphics/obstacles/{choice((0, 1))}.png').convert\_alpha()  
 self.image = pygame.transform.scale(surf, pygame.math.Vector2(surf.get\_size()) \* scale\_factor)  
  
 *# Установка позиции препятствия за пределами экрана* x = WINDOW\_WIDTH + randint(40, 100)  
  
 if orientation == 'up':  
 *# Препятствие снизу* y = WINDOW\_HEIGHT + randint(10, 50)  
 self.rect = self.image.get\_rect(midbottom=(x, y))  
 else:  
 *# Препятствие сверху* y = randint(-50, -10)  
 self.image = pygame.transform.flip(self.image, False, True)  
 self.rect = self.image.get\_rect(midtop=(x, y))  
  
 self.pos = pygame.math.Vector2(self.rect.topleft)  
  
 *# Создание маски для коллизий* self.mask = pygame.mask.from\_surface(self.image)  
  
 def update(self, dt):  
 *# Перемещение препятствия влево* self.pos.x -= 400 \* dt  
 self.rect.x = round(self.pos.x)  
 if self.rect.right <= -100:  
 self.kill() *# Удаление препятствия, когда оно выходит за пределы экрана*

Тестовые примеры работы

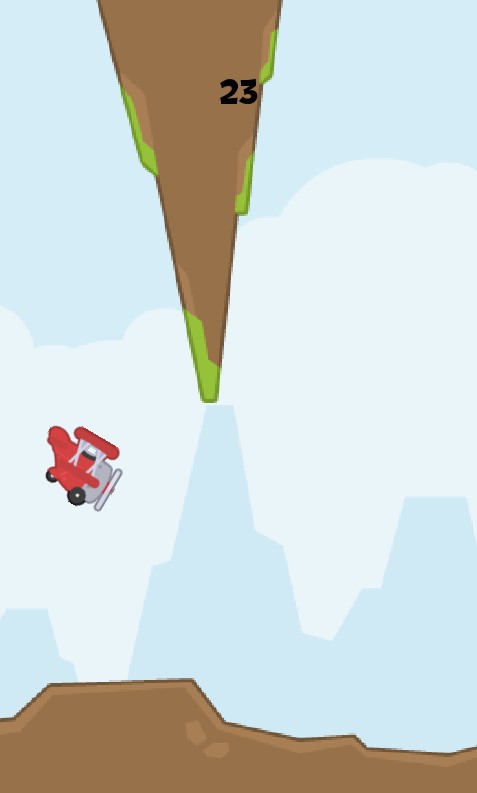


Рисунок 1. Работа программы.

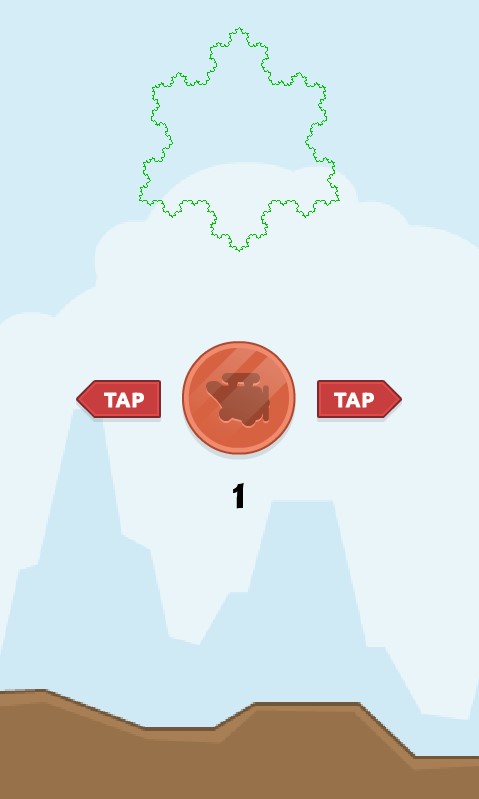


Рисунок 2. Отображение снежинки Коха.

ЗАКЛЮЧЕНИЕ

В рамках данного проекта была реализована аркадная игра с элементами фрактального искусства на языке Python с использованием библиотеки Pygame. Были решены задачи по созданию и управлению объектами игры, обработке коллизий и добавлению звуковых эффектов. Также была внедрена визуализация фрактала (снежинка Коха), который изменяет свою форму в зависимости от времени. Мы создали структурированный код с использованием классов и объектов, что делает проект легко масштабируемым и поддерживаемым. Благодаря использованию спрайтов и масок для обработки столкновений, игра работает эффективно и без задержек. Однако, несмотря на достигнутые результаты, в проекте есть области, которые могут быть улучшены и доработаны в будущем.

Одной из таких областей является оптимизация кода. В текущей реализации можно улучшить производительность, оптимизировав обработку спрайтов и уменьшив нагрузку на процессор при рисовании фрактала. Использование более эффективных алгоритмов и структур данных может значительно повысить FPS (кадры в секунду).

Также возможно расширение функционала игры. Добавление новых типов препятствий и уровней сложности сделает игру более интересной и разнообразной. Например, можно реализовать движущиеся препятствия или изменяющуюся скорость игры. Введение различных бонусов, которые игрок может собирать, добавит элемент стратегии в игру.

Улучшение графики и звука также является важным аспектом. Можно улучшить визуальные эффекты, добавить анимации фона и новые спрайты для самолёта и препятствий. Расширение библиотеки звуковых эффектов и музыки поможет создать более погружающую атмосферу. Можно добавить разные звуки для различных действий, таких как столкновения или сбор бонусов.

2. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Андреева, О. В. Основы алгоритмизации и программирования на языке Python : учебник / О. В. Андреева, О. И. Ремизова. — Москва : Издательский Дом МИСиС, 2022. — 149 c. (Дата обращения: 16.05.2024)
2. Библиотека Pygame: [Электронный ресурс]. https://habr.com/ru/articles/588605/ (Дата обращения: 15.05.2024)
3. Графические ресурсы и звуковые эффекты: [Электронный ресурс]. <https://opengameart.org/> (Дата обращения: 29.04.2024)
4. Основы программирования на Python: [Электронный ресурс]. <https://docs.python.org/3/tutorial/index.html> (Дата обращения: 25.05.2024)