#### Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики информационных технологий

Кафедра вычислительной техники и защиты информации

**Отчет по проекту**

на разработку игры

«Flying Bird»

Руководитель

\_\_\_\_\_\_\_\_\_\_\_\_ Т. В. Абрамова

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Исполнители

студенты группы 23ИБ(б)БАС-1

\_\_\_\_\_\_\_\_\_\_\_\_\_ Зианбетов В.А., Терехов Н.Д., Иванов Д.А., Лапатухин А.С., Михайленко Д.А.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Оренбург 2024 г.

Содержание

[1. Задача разработки программного обеспечения 3](#_Toc8420164)

[1.1 Актуальность разработки и обзор аналогов 3](#_Toc8420165)

[1.2 Описание назначения и возможностей программы 3](#_Toc8420166)

[2. Разработка программного обеспечения 5](#_Toc8420167)

[2.1 Структурная схема ПО 5](#_Toc8420168)

2.2 Функциональная модель IDEF0 5

2.3 Иерархия модулей 6

2.4 Руководство программиста 3

2.5 Разработка пользовательского интерфейса 3

2.6 Руководство пользователя 4

3. Заключение 3

[4. Приложения (листинг программы) 7](#_Toc8420175)

# Задача разработки программного обеспечения

# Актуальность разработки и обзор аналогов

**Игра Flappy Bird, хоть и вышла довольно давно, до сих пор остается актуальной темой для обсуждения. Ее успешность среди игроков объясняется простотой и азартностью геймплея. Игра непроста, именно поэтому игроки постоянно возвращаются к ней, стремясь побить свой собственный или чужой рекорд.**

**Как и многие другие классические игры,** Flappy Bird **до сих пор остается одной из самых популярных и любимых на всем свете. Это объясняется несколькими причинами:**

1. **Простота и уникальность игрового процесса. Вы не найдете ничего похожего на** Flappy Bird **— это уникальный и оригинальный геймплей, который был придуман в 2013 году и до сих пор удивляет, и привлекает людей.**
2. **Универсальность.** Flappy Bird **привлекает не только людей определенного возраста или национальности — это игра, которую могут понимать и наслаждаться люди разных возрастов и культур.**
3. **Разнообразие версий. Существует множество различных версий и вариантов этой игры - от классической версии до современных приложений, что делает** Flappy Bird **доступным для игроков на любом устройстве и платформе.**

**Таким образом, можно с уверенностью сказать, что** Flappy Bird **остается актуальным и любимым многими игроками по всему миру, и вряд ли он утратит свою популярность в ближайшее время.**

# Описание назначения и возможностей программы

Игра предназначена для развития логического, аналитического и стратегического мышления у игрока, а также для развития ловкости. Она помогает развить способность логического мышления, рассуждения и аргументации, принципы дедукции и организационные навыки.

Она помогает улучшению пространственного внимания, развивают способность отслеживать движущиеся объекты, снижают импульсивность.

Видеоигры улучшают способность мозга решать несколько задач одновременно и повышают гибкость ума.

Играя в предложенную игру, вы будете развивать умение смотреть на ситуации с разных сторон, видеть, как положительные, так и отрицательные (и наоборот) моменты. Это очень полезно для развития аналитического, критического мышления и ловкости так часто требующегося в нашей жизни.

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

1.2.1. Функция проверки полученных данных. (Проверка данных позволяет ограничить тип данных или значения, которые можно ввести в ячейку.)

1.2.2. Функция обработки полученных данных. (Обработка данных позволяет представить количественные данные в сжатой форме с тем, чтобы облегчить их понимание)

# Разработка программного обеспечения

* 1. **Структурная схема ПО**

### На рисунке 3 изображена структурно-функциональная схема программного обеспечения.

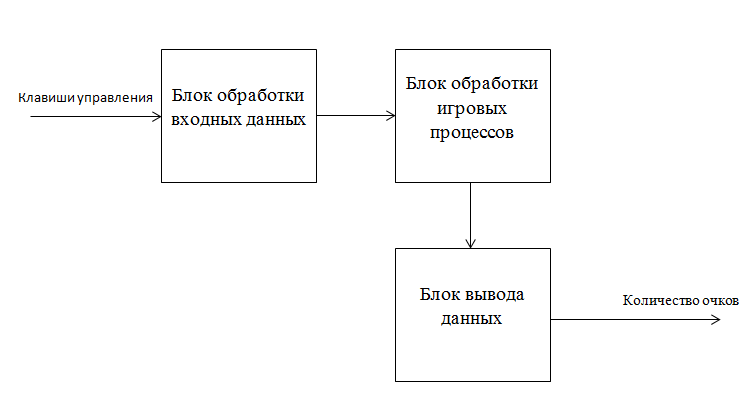


Рисунок 1 – структурно-функциональная схема

Данная схема описывает каждый блок, функцию из которой состоит программа.

# Функциональная модель IDEF0

На рисунке 1 показана функциональная модель IDEF0.

Изображение выглядит как текст, диаграмма, снимок экрана, Параллельный

Автоматически созданное описание

Рисунок 2 – общая схема алгоритма программы

На рисунке 2 изображена подробная схема IDEF0.

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Автоматически созданное описание

Рисунок 3 – подробная схема алгоритма программы

# Иерархия модулей

На рисунке 5 показана схема иерархии модулей.

Рисунок 4 – иерархия модулей

Программа состоит только из одного модуля.

* 1. **Руководство программиста.**

Описание проекта

Игра написана на Python с использованием библиотеки Tkinter для создания графического интерфейса.

Структура кода

Основной класс игры `StupidBirdGame` содержит все методы и атрибуты, необходимые для работы игры.

Основные методы класса:

- `\_\_init\_\_(self, root)`: Инициализация параметров игры, создание виджетов Tkinter, загрузка изображений и GIF-анимации.

- `load\_resized\_gif(self, filepath, size)`: Загрузка и изменение размера всех кадров GIF.

- `show\_start\_screen(self)`: Отображение начального экрана.

- `show\_end\_screen(self)`: Отображение финального экрана.

- `on\_key\_down(self, event)`: Обработка нажатия клавиш.

- `on\_key\_up(self, event)`: Обработка отпускания клавиш.

- `start\_game(self)`: Запуск игры.

- `stop\_update\_game(self)`: Остановка предыдущих вызовов метода `update\_game`.

- `update\_game(self)`: Обновление состояния игры.

- `update\_bird\_animation(self)`: Обновление анимации птички.

- `intersects(self, r1, r2)`: Проверка пересечения двух прямоугольников.

Ресурсы

Для работы игры необходимы следующие ресурсы:

- `start.png`: Изображение начального экрана.

- `end.png`: Изображение экрана окончания игры.

- `bird.gif`: Анимация птички.

- `fon.jpg`: Изображение фона.

- `wall.png`: Изображение нижнего столба.

- `wall up.png`: Изображение верхнего столба.

- `ground.png`: Изображение земли.

* 1. **Разработка пользовательского интерфейса**

Интерфейс программы реализован с помощью библиотек Pycharm. Все текстуры объектов, как и сами объекты были созданы с помощью интернета и программы PhotoShop. Чтобы начать процесс игры нужно нажать клавишу Enter. Процесс игры происходит путем передвижения труб к птице, а сама птица движется только вверх и вниз. Управление птицей происходит путём нажатия на клавишу Space. При нажатии на клавишу Space программа заносит значение 13 в публичную переменную “gravity”, это реализовано с помощью назначенной клавиши в Pycharm. Но при этом существует и ещё одна переменная “gravity”, которая принадлежит назначенной клавиши KeyDawn, которая в свою очередь заносит в свою переменную значение 13. Таким образом есть две команды KeyUp и KeyDawn назначенные на одну клавишу Space. И когда игрок не нажимает клавишу Space в переменной “gravity” значения уменьшаются на 13 и увеличивается на -13, таким образом птица падает вниз. Подсчёт очков происходит, когда расстояние от крайнего верхнего левого элемента трубы до птицы равен сумме ширины трубы и ширины птицы, то есть, когда птица полностью пересекает трубу. Каждая преодолённая труба прибавляет к количеству очков ещё 1. Когда количество очков становиться больше 10 у скорость увеличивается на 1 условную единицу и потом ещё через 10 очков (то есть, когда количество очков преодолеет отметку в 20 очков) скорость увеличится ещё на 1 условную единицу и так далее. Начальная скорость составляет 25 условных единиц. Когда границы объектов (птицы и трубы) пересекаются игра останавливается. Счётчик очков реализован с помощью элемента Label. Так же масштабирование окна игры отключено для корректного отображения всех объектов интерфейса.

На рисунке 5 показан интерфейс начала игры.

Изображение выглядит как текст, снимок экрана, мультфильм, дизайн

Автоматически созданное описание

Рисунок 5 – интерфейс программы в начале игры

На рисунке 6 показан процесс игры.

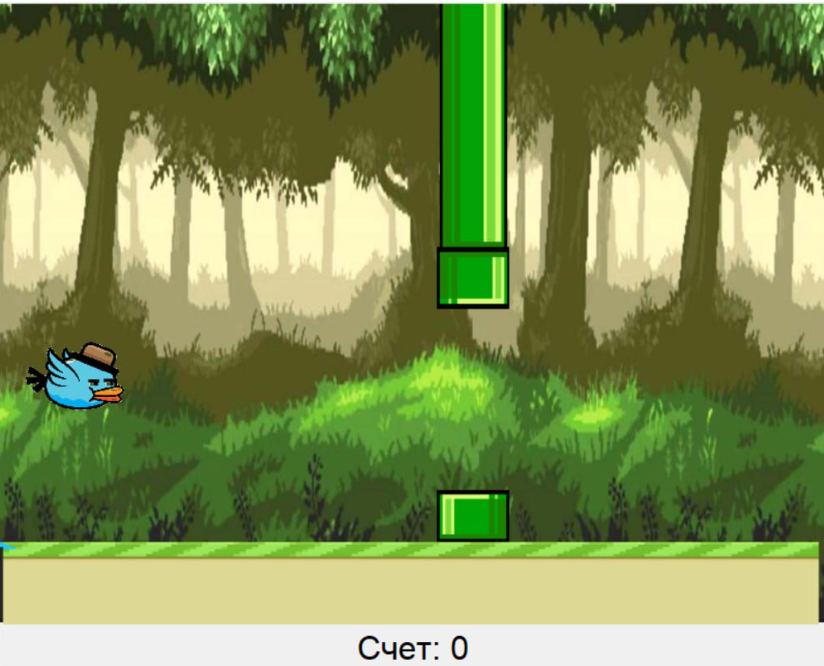


Рисунок 6 – процесс игры

На рисунке 7 показан интерфейс конца игры.

Изображение выглядит как текст, снимок экрана, мультфильм, логотип

Автоматически созданное описание

Рисунок 7 – интерфейс программы в конце игры

* 1. **Руководство пользователя.**

Запуск игры:

- Убедитесь, что у вас установлены библиотеки `tkinter` и `PIL`.

- Поместите все необходимые изображения в одну директорию с файлом скрипта.

- Запустите скрипт: `python game.py`.

Управление:

- Используйте клавишу Space для управления птичкой (подъем и падение).

- Нажмите Enter для начала новой игры после окончания предыдущей.

Цель игры:

- Управляйте птичкой, избегая столкновений со столбами и землей.

- Набирайте очки за каждый успешно пройденный набор столбов.

Завершение игры:

- Игра заканчивается при столкновении птички с препятствием.

- Отображается экран с результатом и предложением начать новую игру (нажмите Enter).

**Заключение**

Игра «Flying Bird» позволяет развивать логическое, аналитическое и стратегическое мышление, а также ловкость. Благодаря простоте и уникальности игрового процесса, она остается актуальной и интересной для игроков всех возрастов.

# Приложение А

*(обязательное)*

**Код программы**

import tkinter as tk  
from PIL import Image, ImageTk, ImageSequence  
import random  
  
  
class StupidBirdGame:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Глупая Птица")  
  
 # Создаем холст для отображения игры  
 self.canvas = tk.Canvas(root, width=800, height=600)  
 self.canvas.pack()  
  
 # Загружаем начальное и финальное изображения  
 self.start\_image = Image.open("start.png")  
 self.start\_image = self.start\_image.resize((800, 400))  
 self.start\_photo = ImageTk.PhotoImage(self.start\_image)  
  
 self.end\_image = Image.open("end.png")  
 self.end\_image = self.end\_image.resize((800, 400))  
 self.end\_photo = ImageTk.PhotoImage(self.end\_image)  
  
 # Загружаем и обрабатываем GIF-анимацию птички  
 self.bird\_imgs = self.load\_resized\_gif("bird.gif", (95, 75))  
 self.bird\_index = 0  
 self.bird\_photo = self.bird\_imgs[self.bird\_index]  
  
 # Загружаем и изменяем размер фона  
 self.bg\_image = Image.open("fon.jpg")  
 self.bg\_image = self.bg\_image.resize((self.bg\_image.width // 4, 600))  
 self.bg\_photo = ImageTk.PhotoImage(self.bg\_image)  
  
 # Загружаем изображение нижнего столба  
 wall\_image = Image.open("wall.png")  
 wall\_image = wall\_image.resize((70, 300))  
 self.wall\_photo = ImageTk.PhotoImage(wall\_image)  
  
 # Загружаем изображение верхнего столба  
 wall\_top\_image = Image.open("wall up.png")  
 wall\_top\_image = wall\_top\_image.resize((70, 350))  
 self.wall\_top\_photo = ImageTk.PhotoImage(wall\_top\_image)  
  
 # Загружаем изображение земли  
 ground\_image = Image.open("ground.png")  
 ground\_image = ground\_image.resize((800, 100))  
 self.ground\_photo = ImageTk.PhotoImage(ground\_image)  
  
 # Инициализируем игровые объекты как None  
 self.bird = None  
 self.wall = None  
 self.wall\_top = None  
 self.ground = None  
 self.bg1 = None # Первый фон  
 self.bg2 = None # Второй фон  
  
 # Устанавливаем начальные значения  
 self.score = 0  
 self.gravity = 10  
 self.initial\_speed = 20 # Начальная скорость  
 self.speed = self.initial\_speed  
 self.score\_threshold = 10  
 self.pipe\_gap = 250  
  
 # Создаем метку для отображения счета  
 self.score\_label = tk.Label(root, text="Счет: 0", font=("Arial", 24))  
 self.score\_label.pack()  
  
 # Привязываем обработчики событий для нажатия клавиш  
 self.root.bind("<KeyPress>", self.on\_key\_down)  
 self.root.bind("<KeyRelease>", self.on\_key\_up)  
  
 # Устанавливаем флаг для проверки состояния игры  
 self.game\_running = False  
 self.resetting = False # Флаг для проверки перезапуска игры  
 self.after\_id = None # ID вызова after  
  
 # Показываем начальный экран  
 self.show\_start\_screen()  
  
 def load\_resized\_gif(self, filepath, size):  
 *"""Загружает GIF и изменяет размер всех кадров"""* img = Image.open(filepath)  
 frames = []  
 for frame in ImageSequence.Iterator(img):  
 frame = frame.resize(size)  
 frames.append(ImageTk.PhotoImage(frame))  
 return frames  
  
 def show\_start\_screen(self):  
 *"""Отображаем начальный экран с инструкцией"""* self.canvas.delete("all")  
 self.start\_label = tk.Label(self.root,  
 text="Нажмите Enter для начала игры\nИспользуйте пробел для управления птичкой",  
 font=("Arial", 24))  
 self.start\_label.pack()  
 self.canvas.create\_image(400, 300, anchor=tk.CENTER, image=self.start\_photo)  
  
 def show\_end\_screen(self):  
 *"""Отображаем финальный экран с результатом"""* self.canvas.delete("all")  
 self.end\_label = tk.Label(self.root, text=f"Счет: {self.score}\nНажмите Enter для новой игры",  
 font=("Arial", 24))  
 self.end\_label.pack()  
 self.canvas.create\_image(400, 300, anchor=tk.CENTER, image=self.end\_photo)  
  
 def on\_key\_down(self, event):  
 *"""Обработчик нажатия клавиш"""* if event.keysym == "space":  
 self.gravity = -13  
 elif event.keysym == "Return":  
 if not self.game\_running and not self.resetting:  
 self.start\_game()  
  
 def on\_key\_up(self, event):  
 *"""Обработчик отпускания клавиш"""* if event.keysym == "space":  
 self.gravity = 13  
  
 def start\_game(self):  
 *"""Запуск игры"""* self.resetting = True # Устанавливаем флаг перезапуска игры  
 if hasattr(self, 'start\_label') and self.start\_label:  
 self.start\_label.destroy()  
 if hasattr(self, 'end\_label') and self.end\_label:  
 self.end\_label.destroy()  
 self.score = 0  
 self.speed = self.initial\_speed # Сбрасываем скорость до начальной  
 self.score\_threshold = 10  
 self.canvas.delete("all")  
  
 # Создаем фон сначала  
 self.bg1 = self.canvas.create\_image(0, 0, anchor=tk.NW, image=self.bg\_photo)  
 self.bg2 = self.canvas.create\_image(self.bg\_image.width, 0, anchor=tk.NW, image=self.bg\_photo)  
  
 # Затем создаем остальные игровые объекты  
 wall\_height = random.randint(200, 400)  
 self.wall = self.canvas.create\_image(600, wall\_height + self.pipe\_gap, anchor=tk.CENTER, image=self.wall\_photo)  
 self.wall\_top = self.canvas.create\_image(600, wall\_height - self.pipe\_gap, anchor=tk.CENTER,  
 image=self.wall\_top\_photo)  
 self.ground = self.canvas.create\_image(400, 620, anchor=tk.S, image=self.ground\_photo)  
  
 self.bird\_index = 0  
 self.bird\_photo = self.bird\_imgs[self.bird\_index]  
 self.bird = self.canvas.create\_image(75, 275, anchor=tk.CENTER, image=self.bird\_photo)  
  
 self.score\_label.config(text="Счет: 0")  
 self.game\_running = True  
 self.resetting = False # Сбрасываем флаг перезапуска игры  
 self.stop\_update\_game() # Останавливаем предыдущие вызовы  
 self.update\_game()  
 self.update\_bird\_animation()  
  
 def stop\_update\_game(self):  
 *"""Останавливаем предыдущие вызовы метода update\_game"""* if self.after\_id is not None:  
 self.root.after\_cancel(self.after\_id)  
 self.after\_id = None  
  
 def update\_game(self):  
 *"""Обновление состояния игры"""* if self.game\_running and not self.resetting:  
 self.canvas.move(self.bird, 0, self.gravity)  
 self.canvas.move(self.wall, -self.speed, 0)  
 self.canvas.move(self.wall\_top, -self.speed, 0)  
  
 self.canvas.move(self.bg1, -self.speed, 0)  
 self.canvas.move(self.bg2, -self.speed, 0)  
  
 # Перемещаем фон обратно, если он выходит за пределы окна  
 if self.canvas.coords(self.bg1)[0] <= -self.bg\_image.width:  
 self.canvas.coords(self.bg1, self.bg\_image.width, 0)  
 if self.canvas.coords(self.bg2)[0] <= -self.bg\_image.width:  
 self.canvas.coords(self.bg2, self.bg\_image.width, 0)  
  
 bird\_coords = self.canvas.bbox(self.bird)  
 wall\_coords = self.canvas.bbox(self.wall)  
 wall\_top\_coords = self.canvas.bbox(self.wall\_top)  
 ground\_coords = self.canvas.coords(self.ground)  
  
 if wall\_coords[2] < 0:  
 x = 800  
 wall\_height = random.randint(200, 400)  
 self.canvas.coords(self.wall, x, wall\_height + self.pipe\_gap)  
 self.canvas.coords(self.wall\_top, x, wall\_height - self.pipe\_gap)  
 self.score += 1  
  
 if (self.intersects(bird\_coords, wall\_coords) or  
 self.intersects(bird\_coords, wall\_top\_coords) or  
 bird\_coords[3] >= ground\_coords[1] - 50):  
 self.game\_running = False  
 self.score\_label.config(text=f"Игра окончена!!!")  
 self.show\_end\_screen()  
 else:  
 self.score\_label.config(text=f"Счет: {self.score}")  
  
 if self.score >= self.score\_threshold:  
 self.speed += 1  
 self.score\_threshold += 10  
  
 if not self.resetting: # Только если игра не перезапускается, вызываем обновление  
 self.after\_id = self.root.after(50, self.update\_game)  
  
 def update\_bird\_animation(self):  
 *"""Обновление анимации птички"""* if self.game\_running:  
 self.bird\_index = (self.bird\_index + 1) % len(self.bird\_imgs)  
 self.bird\_photo = self.bird\_imgs[self.bird\_index]  
 self.canvas.itemconfig(self.bird, image=self.bird\_photo)  
 self.root.after(100, self.update\_bird\_animation) # Скорость анимации (кадры в секунду)  
  
 def intersects(self, r1, r2):  
 *"""Проверка пересечения двух прямоугольников"""* return not (r1[2] < r2[0] or r1[0] > r2[2] or r1[3] < r2[1] or r1[1] > r2[3])  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 root = tk.Tk()  
 game = StupidBirdGame(root)  
 root.mainloop()