Министерство образования Республики Беларусь Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Программирование

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе на тему

**РЕАЛИЗАЦИЯ И ВИЗУАЛИЗАЦИЯ ИГРЫ „FLAPPY BIRD”**

|  |  |
| --- | --- |
| Студент гр.053501 | Р. А. Кривецкий |
| Руководитель: | И. А. Удовин |

2021

*Белорусский государственный университет информатики и радиоэлектроники*

*Факультет компьютерных систем и сетей*

«УТВЕРЖДАЮ»

Заведующий кафедрой информатики

Н. А. Волорова

«\_\_» 2021 года

**ЗАДАНИЕ**

по курсовой работе

*студенту Кривецкому Р.А.*

1. Тема курсовой работы «Реализация и визуализация игры ”Flappy Bird”».
2. Дата защиты курсовой работы « » 2021 г.
3. Исходные данные для курсовой работы
   1. Операционная система Windows.
   2. Язык программирования Java.
4. Содержание пояснительной записки
   1. Введение
   2. Анализ предметной области
   3. Разработка программного средства
   4. Заключение
   5. Список источников
5. Консультант по курсовой работе Удовин И. А.
6. Дата выдачи задания « » 2021 г.

Руководитель курсовой работы *И. А. Удовин*

(подпись)

Задание принял для исполнения *Р. А. Кривецкий*

(дата и подпись студента)

**ОГЛАВЛЕНИЕ**

[Введение 4](#_Toc74136697)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc74136698)

[1.1 Игра „Flappy Bird” 5](#_Toc74136699)

[1.2 Выбор инструментария 5](#_Toc74136700)

[1.3 Постановка задачи 6](#_Toc74136701)

[2. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА 7](#_Toc74136702)

[2.1 Интерфейс 7](#_Toc74136703)

[2.2 Реализация игры 7](#_Toc74136704)

[2.3 Визуализация игры 24](#_Toc74136705)

[3. ЗАКЛЮЧЕНИЕ 27](#_Toc74136706)

[Список источников 28](#_Toc74136707)

# [Введение](#_Введение)

Появление компьютерных игр можно отнести к моменту, когда компьютеры из сферы экспериментальной и почти секретной (ведь на них должны были рассчитываться траектории снарядов и ракет во время военных действий) начали переходить в мир научный и практический. Это произошло в конце 60-х гг. XX в.

Компьютер стал обладать неким более или менее дружественным пользователю интерфейсом - вместо лампочек и загадочных индикаторов появились алфавитно-цифровые дисплеи. Конечно, ни о какой графике не могло идти и речи... Но за компьютерами работают люди, и ничто человеческое им не чуждо. И вот в один прекрасный вечер после тяжёлого трудового дня молодой программист решил написать небольшую программу, которая играла бы с ним в какую-нибудь не очень сложную игру, например «Быки и коровы»... И конечно же, такая идея пришла в голову не только ему одному... Вскоре программы для развлечения начали появляться всё чаще и чаще и даже стали входить в состав пакетов программ, поставляемых вместе с компьютерами.

С созданием компьютерной графики и появлением настоящих домашних компьютеров игровая индустрия стремительно выросла. Игры выпускались тысячными тиражами, не считая пиратских копий.

Сейчас игровая индустрия является одной из точек опоры, на которых стоит индустрия персональных ЭВМ, да и для чего нужен дома компьютер, как не для игр?

В каждой шутке есть доля правды, и мир, наверное, не увидел бы «компьютерной революции», если бы не компьютерные игры.

# [1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ](#_1._АНАЛИЗ_ПРЕДМЕТНОЙ)

## [1.1 Игра „Flappy Bird”](#_1.1_Игра_\„Flappy)

Flappy Bird — игра для мобильных устройств, разработанная вьетнамским разработчиком Донгом Нгуеном, в которой игрок с помощью 9 касаний экрана должен контролировать полёт птицы между рядами зелёных труб, не задевая их. Была реализована на платформах iOS и Android.

Игра вышла 24 мая 2013 года и сразу стала невероятно популярной среди пользователей мобильных устройств по всему миру.

Игровой процесс. Flappy Bird имеет игровой процесс с участием [2D-графики](https://ru.wikipedia.org/wiki/2D-%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0). Цель игры состоит в управлении полётом птицы, которая непрерывно передвигается между рядами зелёных труб. При столкновении с ними происходит завершение игры. Управление производится касанием экрана, при котором птица совершает небольшой рывок вверх. При отсутствии рывков птица падает из-за силы тяжести, и игра также завершается. Очки набираются при каждом успешном перелёте между двумя трубами. Геймплей не имеет изменений на протяжении всей игры.

## [1.2 Выбор инструментария](#_1.1_Выбор_инструментария)

Для реализации курсовой работы была выбрана интегрированная среда разработки JetBrains Intellij Idea для разработки на языке Java. Программное обеспечение JetBrains IntelliJ IDEA – это ведущая среда быстрой разработки на языке Java. Уникальные возможности JetBrains IntelliJ IDEA избавляют программиста от груза рутинной работы, помогают своевременно устранить ошибки и повысить качество кода, поднимая продуктивность разработчика на новую высоту.

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Java задумывался как универсальный язык программирования, который можно применять для различного рода задач. И к настоящему времени язык Java проделал большой путь, было издано множество различных версий. А Java превратилась из просто универсального языка в целую платформу и экосистему, которая объединяет различные технологии, используемые для целого ряда задач: от создания десктопных приложений до написания крупных веб-порталов и сервисов. Достаточно вспомнить популярность мобильной ОС Android, большинство программ для которой пишутся именно на Java.

Также был задействован GitHub — веб-сервис для хостинга IT-проектов и их совместной разработки.

## [1.3 Постановка задачи](#_1.1._Постановка_задачи)

В задачу курсовой работы входит разработка программного средства на языке Java, которое:

* Предоставляет возможность самостоятельно сыграть в игру „Flappy bird”
* Имеет понятный, удобный и красивый пользовательский интерфейс
* Позволяет визуализировать весь игровой процесс

В данной игровой программе игрок будет управлять одним персонажем, в данном случае птичкой, с помощью клавиатуры. Игровой процесс будет составлять продвижение птички, перепрыгивая движущиеся навстречу колонны. Птичка имеет 1 жизнь – 1 попытку, после проигрыша игра начинается с самого начала.

# [2. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА](#_2._РАЗРАБОТКА_ПРОГРАММНОГО)

## [2.1 Интерфейс](#_2.1_Интерфейс)

Для создания легкого и приятного в использовании интерфейса приложения были выделены следующие критерии:

* Интерфейс должен быть удобен в использовании
* Интерфейс должен быть информативным
* Интерфейс должен быть приятным на вид и отвечать всем стандартам разработки современных десктопных приложений

Для реализации данных критериев были использованы графические библиотеки, такие как AWT(Abstract Windows Toolkit) и Swing. AWT использована для создания простых графических объектов, а Swing – для создание оконного приложения и рендеринга анимации игры. Подробнее о данных библиотеках в следующем подразделе.

## [2.2 Реализация игры](#_2.2_Реализация_игры)

В языке Java пакет является специальным библиотечным модулем, который содержит группу классов, объединённых в одном пространстве имён. Для их подключения используется ключевое слово import. Соответственно, для написания программы были использованы следующие пакеты:

* java.awt – содержит все классы для того, чтобы создать пользовательские интерфейсы и для рисования графики и изображений.
* java.awt.Color – обеспечивает классы для цветовых пространств
* java.awt.Font – обеспечивает классы и интерфейсы, касающиеся шрифтов
* java.awt.Graphics –  обеспечивает абстрактный базовый класс для всех графических контекстов, которые позволяют приложению тянуть на компоненты, которые понимаются на различных устройствах, так же как на внеэкранные изображения.
* java.awt.Rectangle – определяет область в координатном пространстве, которое включается Rectangle верхняя левая точка объекта (x,y) в координатном пространстве, его ширине, и его высоте
* java.awt.Image – обеспечивает классы для создания и изменения изображения
* java.awt.event – обеспечивает интерфейсы и классы для того, чтобы иметь дело с различными типами событий, запущенных компонентами AWT.
* java.awt.event.ActionEvent – семантическое событие, которое указывает, что определенное с помощью компонента действие произошло.
* java.awt.event.ActionListener – интерфейс слушателя для того, чтобы получить события действия
* java.awt.event.KeyEvent – событие, которое указывает, что нажатие клавиши произошло в компоненте
* java.awt.event.KeyListener – интерфейс слушателя для того, чтобы получать событие клавиатуры (нажатие клавиш)
* java.util.ArrayList – реализация массива изменяемого размера интерфейса List
* java.util.Random – экземляр этого класса используется для генерации потока псевдослучайных чисел
* java.io.File – класс, позволяющие получить информацию о файле
* java.io.IOException – обеспечивает исключение ввода-вывода
* javax.ImageIO – основной пакет API ввода-вывода изображения
* javax.swing – обеспечивает ряд "легкого веса" компоненты что, до максимального возможного градуса, работа то же самое на всех платформах
* javax.swing.JFrame – добавляет поддержку компонентной архитектуры JFC/Swing. Окно с рамкой „общего назначения”
* javax.swing.Timer – используется для неоднократного выполнения задачи (анимации)
* javax.swing.JPanel – универсальный легкий контейнер для графических интерфейсов

Игра реализуется в классе FlappyBird, который в свою очередь еще реализовывает 2 интерфейса ActionListener, KeyListener. Также реализовано еще 2 класса: Bird и Renderer1. Класс Bird является базовым классом, на который опираются классы графики и логики. Классы FlappyBird и Renderer1 являются классами, которые реализуют графику и содержат кнопки. В классе Bird находятся все необходимые поля и методы для нашей птички.

В классе Renderer1, который расширяет реализацию класса JPanel благодаря ключевому слову extends, переопределяется метод paintComponent. В этом методе вызывается метод repaint() для экземпляра класса FlappyBird flappybird. Эффект вызова repaint() заключается в том, что Swing автоматически очищает графику на панели и выполняет метод paintComponent для перерисовки графики на этой панели.

Исходный код класса Renderer1:

public class Renderer1 extends JPanel  
{  
 @Override  
 protected void paintComponent(Graphics g)  
 {  
 FlappyBird.*flappyBird*.repaint(g);  
 }  
}

Подробнее о классе Bird. Класс имеет целочисленные поля, такие x, y, width, height. За расположение птички, за её координаты отвечают поля x и y. А за размеры птички, за её высоту и ширину отвечают соответственно поля width и height. 2 статические переменные WIDTH и HEIGHT со значениями 750 – это размеры оконного приложения. И поле img класса Image – отвечает за внешний вид птички, которая в свою очередь является изображением. Вызывая конструктор класса Bird() мы присваиваем координатам x, y половину значений WIDTH и HEIGHT, для того, чтобы начальное положение нашей птички было строго по центру графического окна. А размеры птички будут 40 на 40 пикселей, что собственно и показывают поля width и height. Полю img присваиваем картинку, которую считывает метод класса ImageIO.read(new File ("src/flappyBird/bird.png")), если файл будет не найден, то нам возникнет исключение. Метод printStackTrace() отображает сообщение об ошибке в консоли, где мы получаем исключение в исходном коде.

Исходный код класса Bird:

public class Bird  
{  
 public int x, y, width, height;  
 public static int *WIDTH* = 750, *HEIGHT* = 750;  
 private Image img;  
 public Bird() {  
 x = *WIDTH* / 2;  
 y = *HEIGHT* / 2;  
 width = 40;  
 height = 40;  
 try {  
 img = ImageIO.*read*(new File("src/flappyBird/bird.png"));  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public void update(Graphics g) {  
 g.drawImage(img, x, y, width, height, null);  
 }  
  
}

Наша птичка имеет единственный метод update(), в котором будет вызываться метод drawImage(). То есть будет происходить рисование нашей птички. (см. Рисунок 1)

Рисунок 1

Самый главный класс программы – это FlappyBird. Класс реализует интерфейс ActionListener для того, чтобы получать события действия и интерфейс KeyListener для того, чтобы получать события клавиатуры. FlappyBird имеет следующие поля:

* flappyBird – статический экземпляр класса
* WIDTH, HEIGHT – размеры графического окна
* renderer – экземпляр класса Renderer1
* fakeBird – экземпляр класса Rectangle, мнимая птичка
* Bird – экземпляр класса Bird, видимая птичка
* columns – массив объектов Rectangle, колонны
* ticks – целочисленное поле, отвечающее за перевод игры в следующее состояние в зависимости от скорости игры
* yMotion – целочисленное поле, отвечающее за движение по y-координате(прыжок)
* score – поле, отвечающее за счет игры
* gameOver – поле типа boolean, отвечающее за проигрыш
* started – поле типа boolean, отвечающее за старт игры
* rand – экземпляр класс Random для генерации случайных чисел

Исходный код класса FlappyBird:

public class FlappyBird implements ActionListener, KeyListener {  
  
 public static FlappyBird *flappyBird*;  
  
 public final int WIDTH = 750, HEIGHT = 750;  
  
 public Renderer1 renderer;  
  
 public Rectangle fakeBird;  
  
 public Bird bird;  
  
 public ArrayList<Rectangle> columns;  
  
 public int ticks, yMotion;  
  
 public float score;  
  
 public boolean gameOver, started;  
  
 public Random rand;

В конструкторе класс FlappyBird делаем следующее действие:

создаём объекты jframe, timer(с задержкой 17мс), renderer, rand, fakeBird, bird и массив объектов columns.

Для объекта jframe вызываем такие методы, как:

* add(renderer) – используется для добавления визуальных компонентов в кадр
* setTitle(„Flappy Bird”) – устанавливаем название окну „Flappy Bird”
* setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE) – для того, чтобы при закрытии окна закрывалась и программа, иначе она останется висеть в процессах
* setSize(WIDTH, HEIGHT) – устанавливаем размеры окна, равные полям WIGTH на HEIGHT
* addKeyListener(this) – добавляет интерфейс KeyListener для чтения клавиш
* setResizable(false) – для того, чтобы размер окна не мог быть изменен пользователем
* setVisible(true) – определяет, будет ли окно отображаться на экране

После этого 2 раза вызываем метод addColumn(true) и запускаем таймер.

Исходный код конструктора FlappyBird:

public FlappyBird() {  
 JFrame jframe = new JFrame();  
 Timer timer = new Timer(17, this);  
  
 renderer = new Renderer1();  
 rand = new Random();  
  
 jframe.add(renderer);  
 jframe.setTitle("Flappy Bird");  
 jframe.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 jframe.setSize(WIDTH, HEIGHT);  
 jframe.addKeyListener(this);  
 jframe.setResizable(false);  
 jframe.setVisible(true);  
  
 fakeBird = new Rectangle(WIDTH / 2, HEIGHT / 2, 40, 40);  
 bird = new Bird();  
  
 columns = new ArrayList<Rectangle>();  
  
 addColumn(true);  
 addColumn(true);  
 timer.start();  
}

Метод addColumn() отвечает за созданию колонн. В нём создаются целочисленные переменные: space – расстояние между колоннами,

width – ширина колонны, height – высота колонн, генерируемая случайным образом. Внутри условного оператора создаются верхняя и нижняя колонны(добавлением объектов в массив columns) Это вызывается, если игра начата, т.е. start = true. В противном случае следующая реализация создания колонн.

Исходный код метода addColumn :

public void addColumn(boolean start) {  
 int space = 300;  
 int width = 100;  
 int height = 50 + rand.nextInt(300);  
  
 if (start) {  
 columns.add(new Rectangle(WIDTH + width + columns.size() \* 300, HEIGHT - height - 120, width, height));  
 columns.add(new Rectangle(WIDTH + width + (columns.size() - 1) \* 300, 0, width, HEIGHT - height - space));  
 } else {  
 columns.add(new Rectangle(columns.get(columns.size() - 1).x + 600, HEIGHT - height - 120, width, height));  
 columns.add(new Rectangle(columns.get(columns.size() - 1).x, 0, width, HEIGHT - height - space));  
 }  
}

Метод paintColumn() отвечает за отрисовку колонн в виде прямоугольников. В нём вызываются методы setColor() для установки цвета колонн, в данном случае зеленый и fillRect() для закрашивания прямоугольников.

Исходный код метода paintColumn:

public void paintColumn(Graphics g, Rectangle column) {  
 g.setColor(Color.*green*.darker());  
 g.fillRect(column.x, column.y, column.width, column.height);  
}

Один из самых главных методов программы является jump(). Он отвечает за логику и физику прыжка птички. При условии, что игра окончена(gameOver = true) создаются объекты классов fakeBird, который будет находится в центре окна и размером 40 на 40 пикселей и Bird. Также для массива columns вызывается метод clear(), чтобы очистить окно от колонн с прошлой игры. Естественно аннулируются счетчик очков score и

изменение y-координаты yMotion. Также 2 раза вызываем метод addColumn(). Сам прыжок заключается в переменной yMotion, уменьшая её значение на 10, тем самым производим прыжок птички на 10 пикселей. Также добавляем условие, что если yMotion становится положительным числом, значение аннулируем.

Исходный код метода jump:

public void jump() {  
 if (gameOver) {  
  
 fakeBird = new Rectangle(WIDTH / 2, HEIGHT / 2, 40, 40);  
 bird = new Bird();  
 columns.clear();  
 yMotion = 0;  
 score = 0;  
  
 addColumn(true);  
 addColumn(true);  
  
 gameOver = false;  
 }  
 if (!started) {  
 started = true;  
 } else if (!gameOver) {  
 if (yMotion > 0) {  
 yMotion = 0;  
 }  
  
 yMotion -= 10;  
 }  
}

Метод actionPerfomed(ActionEvent e) вызывается, когда пользователь нажмет кнопку, объект - кнопка создаст объект класса ActionEvent. Реализация этого метода - действия, происходящие при наступлении события. В данном методе реализуются самые ключевые моменты игры: движение колонн, перемещение птички, условия проигрыша, перерисовка и рендеринг. Переменная speed отвечает за скорость движения колонн. Как происходит повышение счетчика: если координаты птички bird попадают между колоннами. Тогда счетчик повышается на единицу. Проигрыш происходил в случае, когда птичка касалась верхней либо нижней границ окна. Проверяется по координатам. Но также проигрыш засчитывается, когда птичка коснётся колонн. Как именно это реализовано? Один из самых интересных моментов программы. Именно поэтому и было создано 2 объекта для птички: fakeBird, Bird. Первая отвечает как раз за внутреннюю логику, а именно за проигрыш. Объект fakeBird является экземпляром класса Rectangle, как и колонны. Соответственно, появилась возможность вызвать метод intersects(). Его суть заключается в том, что пересекаются ли объекты формы, в данном случае прямоугольников. Как только происходит пересечение column и fakeBird, gameOver = true, т.е. игра окончена. А объект Bird отвечает за внешнюю – визуальную составляющую птички. Чтобы не происходило „осечек”, квадратная fakeBird и картинка bird имеет одинаковые размеры, одинаковые координаты всегда, т.е. движутся неотрывно друг от друга. Только квадрат fakeBird не виден пользователю, а картинку bird можем видеть. И естественно, когда координаты птички совпадают с верхней или нижней границами окна, игра оканчивается. И последнее в конце метода actionPerfomed для объекта renderer вызывается метода repaint для перерисовки окна.

Исходный код метода actionPerfomed:

public void actionPerformed(ActionEvent e) {  
 int speed = 10;  
 ticks++;  
  
 if (started) {  
 for (int i = 0; i < columns.size(); i++) {  
 Rectangle column = columns.get(i);  
 column.x -= speed;  
 }  
  
 if (ticks % 2 == 0 && yMotion < 15) {  
 yMotion += 2;  
 }  
  
 for (int i = 0; i < columns.size(); i++) {  
 Rectangle column = columns.get(i);  
  
 if (column.x + column.width < 0) {  
 columns.remove(column);  
  
 if (column.y == 0) {  
 addColumn(false);  
 }  
 }  
 }  
  
 fakeBird.y += yMotion;  
 bird.y += yMotion;  
  
 for (Rectangle column : columns) {  
 if (column.y == 0 && bird.x + bird.width / 2 > column.x + column.width / 2 - 10 && bird.x + bird.width / 2 < column.x + column.width / 2 + 10) {  
 if (gameOver) {  
 break;  
 }  
 score += 0.5;  
 }  
  
 if (column.intersects(fakeBird)) {  
 gameOver = true;  
  
 if (bird.x <= column.x) {  
 bird.x = column.x - bird.width;  
 } else {  
 if (column.y != 0) {  
 bird.y = column.y - bird.height;  
 } else if (bird.y < column.height) {  
 bird.y = column.height;  
 }  
 }  
 }  
 }  
  
 if (bird.y > HEIGHT - 120 || bird.y < 0) {  
 gameOver = true;  
 }  
  
 if (bird.y + yMotion >= HEIGHT - 120) {  
 bird.y = HEIGHT - 120 - bird.height;  
 gameOver = true;  
 }  
 }  
  
 renderer.repaint();  
}

Метод repaint содержит всю необходимую реализацию для перерисовки всех объектов окна, то есть всей визуальной части программы. Первым [отрисовывает](https://m.translate.ru/%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BA%D1%81%D1%82%D1%8B/%D1%80%D1%83%D1%81%D1%81%D0%BA%D0%B8%D0%B9-%D0%BD%D0%B5%D0%BC%D0%B5%D1%86%D0%BA%D0%B8%D0%B9/%D0%BE%D1%82%D1%80%D0%B8%D1%81%D0%BE%D0%B2%D1%8B%D0%B2%D0%B0%D0%B5%D1%82)ся фон цветом cyan(голубой). Это прямоугольник, который заполняет всё окно. Затем отрисовываются колонны цветом green.

Оставляем закомментированной строку, в которой отрисовывалась бы мнимая птичка fakeBird, которая в данном случае является красным квадратом. Но вместо неё появляется изображение bird, оно будет находится в окне до тех пор, пока игрок не проиграет. Как только произойдет gameOver, птичка исчезает. Также в методе происходит отображение главных надписей, поэтому устанавливаем шрифт и белый цвет для этого. Пока игра не начата, на окне находится надпись „Click to start!”, во время игры ведётся отображение счёта в верхней части окна. И при поражении отображаются надписи „Game Over!”, а под ней „Score:” и значение очков, которое набрал игрок.

Исходный код метода repaint:

public void repaint(Graphics g) {  
  
 g.setColor(Color.*cyan*);  
 g.fillRect(0, 0, WIDTH, HEIGHT);  
  
 g.setColor(Color.*green*);  
 g.fillRect(0, HEIGHT - 120, WIDTH, 120);  
  
 g.setColor(Color.*green*);  
 g.fillRect(0, HEIGHT - 120, WIDTH, 20);  
  
 g.setColor(Color.*red*);  
 *//g.fillRect(fakeBird.x, fakeBird.y, fakeBird.width, fakeBird.height);* if (!gameOver && started)  
 bird.update(g);  
  
 for (Rectangle column : columns) {  
 paintColumn(g, column);  
 }  
  
 g.setColor(Color.*white*);  
 g.setFont(new Font("Arial", 1, 100));  
  
 if (!started) {  
 g.drawString("Click to start!", 75, HEIGHT / 2 - 50);  
 }  
  
 if (gameOver) {  
 g.drawString("Game Over!", 100, HEIGHT / 2 - 50);  
 g.drawString("Score: " + (int)score, 200, HEIGHT / 2 + 75);  
  
 }  
  
 if (!gameOver && started) {  
 g.drawString(String.*valueOf*((int)score), WIDTH / 2 - 25, 100);  
 }  
  
}

Методы, отвечающие за нажатие клавиш. Все ключевые события для прослушивателя ключей, потому что это просто необходимо для правильной компиляции программы в Java. Поэтому метод keyReleased отвечает за того, что когда нажат Space – вызывается метод jump, т.е. происходит прыжок птички. А если во время игры игрок хочет в данный момент завершить игру, то при нажатии Escape – игра завершается с тем счётом, который был. 2 других метода keyTyped и keyPressed не имеют никакой реализации, но без них не работает интерфейс KeyListener.

Исходный код методов keyReleased, keyTyped, keyPressed:

public void keyReleased(KeyEvent e) {  
 if (e.getKeyCode() == KeyEvent.*VK\_SPACE*) {  
 jump();  
 }  
 if (e.getKeyCode() == KeyEvent.*VK\_ESCAPE*) {  
 gameOver = true;  
 }  
}  
  
public void keyTyped(KeyEvent e) {  
  
}  
  
public void keyPressed(KeyEvent e) {  
  
}

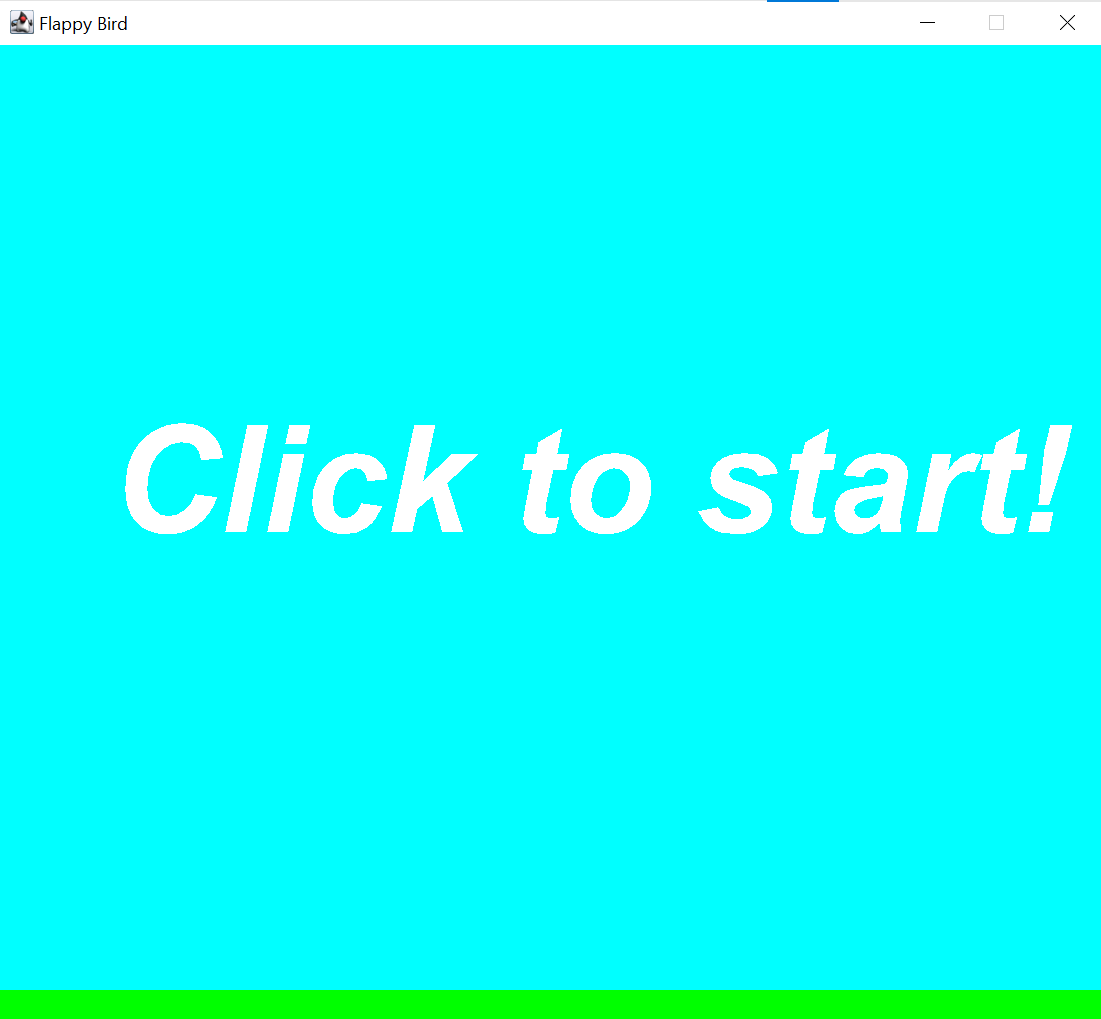
И самое последнее: вызов метода main для запуска самой игры. В нём создается объект flappyBird.

Исходный код main:

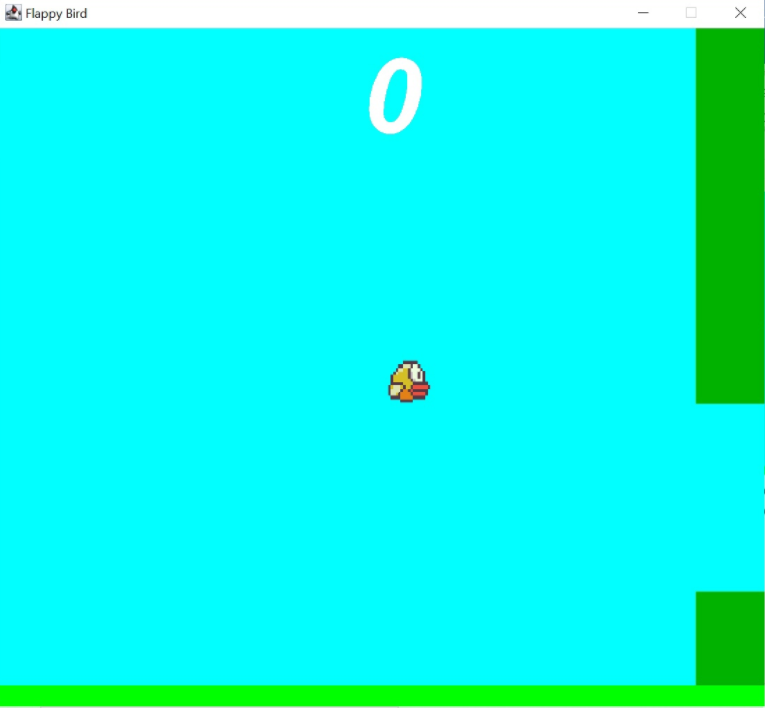
public static void main(String[] args) {  
 *flappyBird* = new FlappyBird();  
}

## [2.3 Визуализация игры](#_2.3_Визуализация_игры)

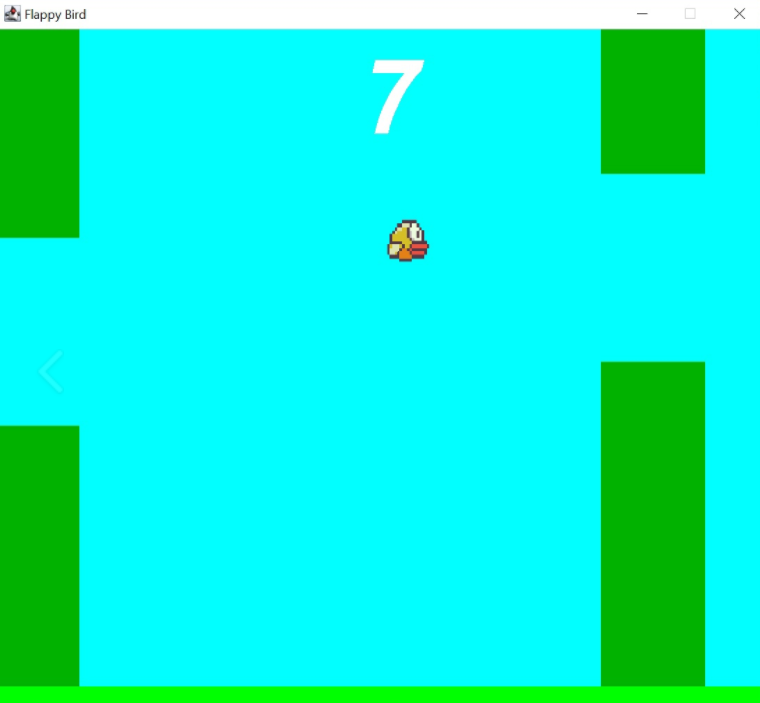
Когда уже весь код написан, ничего не остаётся кроме, как запустить программу и оценить полученный продукт.

 Рисунок 2

Первоначальное окно отображается на рисунке 2. Игра еще не начата, птички и колонны еще не появились.

 Рисунок 3

Сразу после нажатия Space запускается игровой процесс, первый прыжок птички отображается на рисунке 3.

 Рисунок 4

На рисунке 4 отображается уже полностью весь игровой процесс, птичка уже прошла несколько препятствий, счёт равен 7.

 Рисунок 5

На рисунке 7 отображается конечное окно. Оно возникает, когда игрок проиграл. На окне появляется счёт за данную игру, в данном случае игрок набрал 10 очков.

# [3. ЗАКЛЮЧЕНИЕ](#_3._ЗАКЛЮЧЕНИЕ)

В результате выполнения курсовой работы было разработано приложение, являющееся полноценным аналогом игры „Flappy Bird”, пусть и с упрощенной графикой, но с той же логикой и игровым процессом. Программа может быть модернизирована и дополнена. Возможны увеличение сложности (ускорить приближение колонн, уменьшение расстояния между труб), улучшение анимации и игровой графики в целом, добавление звукового сопровождения, создание меню опций с такими пунктами, как полноэкранный/оконный режим игры, изменение уровня сложности и др. Это вызвало бы больший интерес к игре. Приложение представляет собой очень простой вариант игры. При работе постоянно возникали сложности, которые решались по мере написания кода. После завершения проекта были сделаны следующие выводы:

* Объектно-ориентированное программирование является отличным способом разработки как крупных, так и малых проектов. Имеет множество преимуществ над процедурно-ориентированными.
* Представление архитектуры проекта и взаимодействий компонента является не менее важной частью разработки, чем написание кода.

Разработанный программный продукт может считаться игровой программой и удовлетворять требованиям пользователя. Целевой аудиторией данной игровой программы предполагаются любители игрового жанра „Аркада”, который требует напряжения внимания и быстрой реакции на происходящие в игре события.

В процессе кодирования была изучена Java в части пользовательского интерфейса и технологий отображения графики. В процессе тестирования приложения ошибок не обнаружено.

# 

# [Список источников](#_Список_источников)

1. Wikipedia: <https://ru.wikipedia.org/wiki/Flappy_Bird>
2. Metanit: <https://metanit.com/java/tutorial/1.1.php>
3. I.T.Pro: <https://itpro.ua/product/jetbrains-intellij-idea/?tab=description>
4. Habr: https://habr.com/ru/post/243471/
5. https://spec-zone.ru/RU/Java/Docs/7/api/overview-summary.html