A

Minor Project Report

On

**Flappy Bird: Simple windows game with challenges**

Submitted To

**Bhilai Institute of Technology, Durg**

*in partial fulfillment of requirement for the award of degree*

of

Bachelor of Technology

in

Information Technology

By

**Arushi Agrawal**

**Subhodeep Sarkar**

**Shubham Vishwakarma**

|  |  |
| --- | --- |
|  | **DEPARTMENT OF INFORMATION TECHNOLOGY**  **BHILAI INSTITUTE OF TECHNOLOGY**  **BHILAI, DURG, CHHATTISGARH** |

Session: 2022-2023

**DECLARATION**

We the undersigned solemnly declare that the report of the project work entitled “**Flappy Bird:Simple windows game with challenges”** is based on our own work carried out during the course of our study under the supervision of **Mr. P.D Mishra**.

We assert that the statements made and conclusions drawn are an outcome of the project work. We further declare that to the best of our knowledge and belief that the report does not contain any work which has been submitted for the award of any other degree/diploma/certificate in this university or any other university.

(Signature of the Candidate)

Name of the candidate: Arushi Agrawal

Roll No.-300103321022

Enrolment No.-CA6643

(Signature of the Candidate)

Name of the candidate: Subhodeep Sarkar

Roll No.-300103321050

Enrollment No.-CB4397

(Signature of the Candidate)

Name of the candidate: Shubham Vishwakarma

Roll No.-300103321051

Enrollment No.-CB4409

**CERTIFICATE**

This is to certify that the report of the mini project submitted is an outcome of the project work entitled “**Flappy Bird:Simple windows game with challenges”** carried out by **Arushi Agrawal** Roll No: 300103321022 Enrolment No. CA6643**, Subhodeep Sarkar** Roll No: 300103321050 Enrolment No. CB4397 and **Shubham Vishwakarma** Roll No: 300103321051 Enrolment No. CB4399 under my guidance and supervision for the award of Degree in Bachelor of Engineering in Information Technology of Bhilai Institute of Technology, Durg (C.G) , India.

To the best of my knowledge the report

* Embodies the work of the candidate himself ,
* Has duly been completed,
* Fulfils the requirement of the Ordinance relating to the B.E. degree of the University,
* Is up to the desired standard for the purpose of which is submitted.

(Signature of the Guide)

Name of Guide: Mr. P.D Mishra

Designation: Assistant Professor

Department: Information Technology

Name and Address of the Institute: B.I.T. Durg

The project work as mentioned above is hereby being recommended and forwarded for examination and evaluation.

(Signature of Head of Dept. with seal)

**CERTIFICATE BY THE EXAMINERS**

This is to certify that the project work entitled “**Flappy Bird:Simple windows game with challenges”**. Submitted by **Arushi Agrawal** Roll No: 300103321022 Enrolment No. CA6643**, Subhodeep Sarkar** Roll No: 300103321050 Enrolment No. CB4397 and **Shubham Vishwakarma** Roll No: 300103321051 Enrolment No. CB4399 has been examined by the undersigned as a part of the examination for the award of Bachelor of Technology degree in Information Technology.

Internal Examiner External Examiner

Date: Date:

**ACKNOWLEDGEMENT**

Any accomplishment requires the efforts of several people, and this work is no different. We would take this opportunity to extend our sincerest gratefulness towards our Head of the Department **Dr. Ani Thomas** and our guide/project-in-charge **Mr. P.D Mishra** for their valuable time and suggestions towards our project, without which this project would not have been a success today.

(Signature of Student) (Signature of Student) (Signature of Student)

Name: Arushi Agrawal Name: Subhodeep Sarkar Name: Shubham Vishwakarma

Roll No. – 300103321022 Roll No. -300103321050 Roll No.- 3000103321051

Enrolment No. - CA6643 Enrolment No. - CB4397 Enrolment No. – CB4399

**TABLE OF CONTENTS**

|  |
| --- |
|  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter** | **Chapter Title** | **Section Title** | **Page no.** |
|  | **ABSTRACT** | Abstract | 7 |
| **I** | **Introduction** | Overview | 8 |
| Game play mechanics | 8 |
| Benefits of Flappy Bird game | 8 - 9 |
| **II** | **PROBLEM IDENTIFICATION** | Problem Statement | 9 |
| **III** | **METHODOLOGY** | Methodology | 9 - 11 |
| Hardware and Software used | 11 - 15 |
| Flappy Bird code | 15 - 32 |
| Code explanation | 33 |
| Gameplay screenshots | 34 - 35 |
| **IV** | **RESULTS AND DISCUSSIONS** | Results And Discussion | 36 |
| **V** | **CONCLUSION AND SCOPE OF FURTHER WORK** | Conclusion and scope of further work | 36 - 37 |
| **VI** | **BIBLIOGRAPHY** | Bibliography | 38 |

**ABSTRACT**

Flappy Bird is a widely recognized mobile game released in 2013, developed by Dong Nguyen. Its popularity soared due to its straightforward yet challenging gameplay mechanics. In the game, players control a bird by tapping the screen, making it flap its wings to navigate through a series of pipes. The objective is to accumulate the highest score possible by successfully passing through as many pipes as feasible without colliding with them.

Flappy Bird stands as a significant exemplar of the influence a straightforward yet engaging game can wield on popular culture. Its meteoric rise to fame, followed by its sudden removal, prompted conversations about the role of mobile gaming and the ethical obligations of developers. The legacy of Flappy Bird serves as a reminder of the ever-evolving nature of the gaming industry and its capacity to captivate global audiences.

**INTRODUCTION:**

**Overview:**

Flappy Bird quickly gained notoriety after its launch, ascending to the top of app store charts worldwide. Its success can be attributed to its accessible mechanics, rendering it appealing to a broad audience. The game's minimalist design and demanding gameplay contributed to its addictiveness, leading to widespread engagement.

**Gameplay Mechanics**

The central gameplay mechanic of Flappy Bird revolves around tapping the screen to induce the bird to flap its wings, allowing it to ascend. The bird descends due to gravity, necessitating precise timing to maneuver through the narrow gaps between pipes. A player's score is determined by the number of pipes successfully navigated. Any Iot based device consists of

**Benefits of playing Flappy Bird game:**

Playing Flappy Bird, like any game, can have both positive and negative effects. Here are some potential benefits of playing Flappy Bird:

**Hand-Eye Coordination:** Flappy Bird requires precise timing and coordination between the player's taps and the bird's movements. This can help improve hand-eye coordination, which is useful in various real-world activities.

**Focus and Concentration:** The game demands a high level of concentration to navigate the bird through the obstacles. This can help improve focus and attention to detail.

**Reaction Time:** Players need to react quickly to changes in the game environment, such as the bird's altitude and the position of the pipes. This can help improve reaction time, which is valuable in many situations.

**Patience and Perseverance:** Flappy Bird is notoriously challenging, and achieving a high score requires patience and perseverance. Players may develop a greater tolerance for frustration and a willingness to keep trying in the face of difficulty.

**Stress Relief:** Engaging in gaming can serve as a form of relaxation and stress relief for many individuals. It provides an escape from the demands of daily life.

**Entertainment and Enjoyment:** Games are designed to be enjoyable, and playing Flappy Bird can be a source of entertainment and amusement.

**Competition and Achievement:** Setting and achieving high scores in Flappy Bird can provide a sense of accomplishment and competition, which can be motivating for some players.

**Social Interaction:** Although Flappy Bird is primarily a single-player game, players may share their achievements with friends or participate in online communities related to the game.

**Problem statement:**

"Develop a user-friendly and visually appealing mobile game inspired by the mechanics of Flappy Bird, with enhanced features and customization options, to engage and entertain a diverse audience while maintaining a balance between accessibility and challenge."

In this hypothetical scenario, the "problem" being addressed is the desire to create a new game that captures the essence of Flappy Bird while introducing additional features and customization to cater to a wider range of players. The goal is to strike a balance between being accessible to casual players and providing enough challenge to keep them engaged.

**METHODOLOGY:**

Creating a Flappy Bird game using LWJGL (Lightweight Java Game Library) involves several key steps. Below is a high-level methodology to guide you through the process:

**1.Setting Up the Development Environment:**

- Download and install LWJGL, which includes the necessary libraries for graphics, input handling, and window management.

**2. Initializing the Game Window:**

- Use LWJGL to create a window for the game. This involves setting up the display mode, creating a window context, and handling user input events.

**3. Loading Textures and Assets:**

- Load image files for the game elements (bird, pipes, background, etc.) using LWJGL's texture loading capabilities.

**4. Creating the Bird Class:**

- Implement a class to represent the bird character. This class will handle the bird's position, velocity, gravity, and rendering.

**5. Implementing Pipe Generation:**

- Create a class to manage the generation and movement of pipes. Determine the spacing, gap size, and randomization for generating pipes.

**6. Handling Input:**

- Utilize LWJGL's input handling to detect user interactions, such as key presses, to control the bird's movement (e.g., making it jump).

**7. Rendering the Game Scene:**

- Use OpenGL, which is integrated with LWJGL, to render the game elements. This includes the bird, pipes, background, and any other visual components.

**8. Implementing Collision Detection:**

- Define algorithms to detect collisions between the bird and the pipes. This is crucial for determining whether the game should continue or end.

**9. Managing Game State:**

- Implement a state management system to handle game states such as start screen, playing, and game over. This helps control the flow of the game.

**10. Scoring and Game Over Conditions:**

- Track the player's score based on the number of pipes successfully passed. Implement conditions for ending the game, such as collisions or reaching a certain score threshold.

**11. Testing and Debugging:**

- Test the game thoroughly to identify and fix any bugs or issues. Ensure that the game functions as expected under different scenarios.

**HARDWARE & SOFTWARE REQUIREMENTS:**

1. Software Required:
2. Eclipse 4.28 IDE
3. LWJGL library
4. Windows 10 or above OS
5. Hardware used:
   1. RAM: 4GB above
   2. Disk Size: 100MB
   3. Processor: intel i3 or above

**Java:**

Java is a widely-used, versatile, and platform-independent programming language known for its "Write Once, Run Anywhere" (WORA) philosophy. Developed by James Gosling and Mike Sheridan at Sun Microsystems (now owned by Oracle Corporation), Java was first released in 1995.

Here are some key characteristics and features of Java:

**1. Platform Independence:** Java is designed to be platform-independent, meaning that Java code can be executed on any device or operating system that has a Java Virtual Machine (JVM) installed.

**2. Object-Oriented:** Java is an object-oriented programming (OOP) language, which means it organizes code into classes and objects. This promotes modularity, reusability, and maintainability.

**3. Syntax Similar to C/C++:** Java's syntax is similar to that of C and C++, making it relatively easy for developers familiar with these languages to transition to Java.

**4. Automatic Memory Management (Garbage Collection):** Java manages memory automatically, which means developers don't need to explicitly allocate and deallocate memory. The garbage collector automatically reclaims memory that is no longer in use.

**5. Multi-Threading:** Java provides built-in support for multi-threading, allowing programs to execute multiple tasks concurrently. This is essential for developing applications that require parallel processing.

**6. Extensive Standard Library:** Java comes with a large standard library that provides pre-built classes and methods for common tasks like I/O, networking, data structures, and more.

**7. Strong Type Checking:** Java is a statically-typed language, which means variable types must be declared before they are used. This helps catch type-related errors at compile time.

**8. Security:** Java was designed with security in mind. It includes features like classloaders, a security manager, and a bytecode verifier to protect against malicious code.

**9. Rich Ecosystem:** Java has a vast ecosystem of tools, frameworks, and libraries that facilitate development across a wide range of domains, including web development, mobile app development, enterprise software, and more.

**10. Enterprise-Level Scalability:** Java is a popular choice for building large-scale, enterprise-level applications. It provides features like Enterprise JavaBeans (EJBs) and Java EE (Enterprise Edition) for building robust and scalable server-side applications.

**11. Community Support:** Java has a large and active community of developers. This community contributes to the wealth of resources available, including forums, blogs, tutorials, and open-source projects.

**12. Continuous Evolution:** Java is a language that continues to evolve with regular updates and new versions. The Java Community Process (JCP) oversees the development of Java specifications and APIs.

Java's versatility and widespread use make it a popular choice for a wide range of applications, from web and mobile development to enterprise-level software, scientific computing, and more. It remains one of the most widely-used programming languages in the world.

**Eclipse 4.28:**

Eclipse is a widely used Integrated Development Environment (IDE) for Java programming. It provides a robust set of tools and features to assist developers in writing, testing, and deploying Java applications. Here are some key features and aspects of Eclipse IDE for Java:

**1. Platform Independence:** Eclipse is a cross-platform IDE, which means it can run on various operating systems including Windows, macOS, and Linux.

**2. Rich Ecosystem:** Eclipse has a large and active community, which has contributed to a vast ecosystem of plugins and extensions. This allows developers to customize and extend the IDE's functionality to suit their specific needs.

**3. Java Development Tools (JDT):** Eclipse's Java Development Tools provide comprehensive support for Java development. It includes features like code completion, refactoring, syntax highlighting, debugging, and more.

**4. Integrated Debugger:** Eclipse comes with a powerful debugger that allows developers to set breakpoints, inspect variables, step through code, and analyze the program's execution flow.

**5. Version Control Integration:** Eclipse supports version control systems like Git, Subversion (SVN), and CVS, making it easy to manage and collaborate on projects with multiple developers.

**6. Maven and Ant Support:** Eclipse integrates with build automation tools like Maven and Ant, enabling developers to manage project dependencies and build processes seamlessly.

**7. Code Templates and Snippets:** Eclipse provides code templates and snippets that can help developers write code faster and adhere to best practices.

**8. Graphical User Interface (GUI) Builder:** Eclipse offers tools like WindowBuilder that allow for the creation of graphical user interfaces for Java applications through a drag-and-drop interface.

**9. Unit Testing Integration:** Eclipse supports popular unit testing frameworks like JUnit, allowing developers to write and run tests within the IDE.

**10. Code Analysis and Metrics:** Eclipse can perform static code analysis and provide metrics on code quality, which can help developers identify potential issues and maintain code consistency.

**11. Task Management:** Eclipse includes a task management system that allows developers to create and manage tasks, making it easier to keep track of work to be done.

**12. Support for Other Languages:** While Eclipse is primarily known for Java development, it can be extended to support other programming languages through plugins. For example, Eclipse can be used for C/C++ development with the Eclipse CDT plugin.

**13. Marketplace:** Eclipse has a marketplace where users can find and install various plugins, themes, and extensions to enhance their development experience.

Overall, Eclipse is a versatile and powerful IDE that provides a comprehensive set of tools for Java development, making it a popular choice among developers worldwide.

**LWJGL:**

The Lightweight Java Game Library (LWJGL) is a Java library that provides access to native libraries for various aspects of game development, including graphics, audio, input handling, and more. It allows Java developers to create high-performance games and multimedia applications by interfacing with system-level libraries.

Here are some key components and features of LWJGL:

**1. OpenGL Integration:** LWJGL provides bindings for OpenGL, a widely-used graphics API, allowing developers to harness the power of modern graphics hardware for 2D and 3D rendering.

**2. OpenAL Integration:** LWJGL includes bindings for OpenAL, a cross-platform audio API, enabling developers to implement advanced audio features in their games or applications.

**3. LWJGL Display:** It provides a windowing system for creating and managing windows, as well as handling input events like keyboard and mouse interactions.

**4. Lightweight and Efficient:** As the name suggests, LWJGL is designed to be lightweight and efficient. It provides direct access to native libraries, avoiding unnecessary overhead.

**5. Platform Independence:** LWJGL is designed to work on various operating systems, including Windows, macOS, and Linux, making it suitable for cross-platform game development.

**6. Support for Multi-threading:** LWJGL allows for multi-threaded programming, which is crucial for handling complex game logic, rendering, and input processing simultaneously.

**7. Flexible and Extensible:** LWJGL is designed to be flexible and extensible, allowing developers to integrate additional libraries and frameworks as needed for their specific projects.

**8. Community and Documentation:** LWJGL has an active and supportive community, with a wealth of documentation, tutorials, and forums available to help developers get started and troubleshoot issues.

**9. Support for Advanced Features:** LWJGL enables access to advanced features such as shaders, framebuffers, vertex buffers, and other modern graphics techniques, allowing for high-quality graphics rendering.

**10. Integration with Other Libraries:** LWJGL can be combined with other Java libraries and frameworks, such as the Lightweight Java Game Library (LWJGL) Utility Library (LWJGLUtil), to streamline game development tasks.

**11. Gamepad and Controller Support:** LWJGL includes support for gamepad and controller input, making it suitable for games that require joystick or gamepad interaction.

**12. Native Code Access:** While LWJGL itself is written in Java, it provides a bridge to native code, allowing developers to access low-level system functions if needed for performance-critical tasks.

Overall, LWJGL is a powerful tool for Java game developers who want to create high-performance, cross-platform games with access to advanced graphics and audio capabilities. It is widely used in the game development community and has been the foundation for many successful Java games and applications.

**Flappy Bird code:**

package com.project.flappy;

import static org.lwjgl.glfw.GLFW.\*;

import static org.lwjgl.opengl.GL11.\*;

import static org.lwjgl.opengl.GL13.\*;

import static org.lwjgl.system.MemoryUtil.\*;

import org.lwjgl.glfw.GLFWVidMode;

import org.lwjgl.opengl.GL;

import com.project.flappy.graphics.Shader;

import com.project.flappy.input.Input;

import com.project.flappy.level.Level;

import com.project.flappy.maths.Matrix4f;

public class Main implements Runnable {

private int width = 1280;

private int height = 720;

private Thread thread;

private boolean running = false;

private long window;

private Level level;

public void start() {

running = true;

thread = new Thread(this, "Game");

thread.start();

}

private void init() {

if (!*glfwInit*()) {

System.***err***.println("Could not initialize GLFW!");

return;

}

*glfwDefaultWindowHints*();

*glfwWindowHint*(***GLFW\_RESIZABLE***, ***GL\_TRUE***);

*glfwWindowHint*(***GLFW\_CONTEXT\_VERSION\_MAJOR***, 4);

*glfwWindowHint*(***GLFW\_CONTEXT\_VERSION\_MINOR***, 6);

*glfwWindowHint*(***GLFW\_RESIZABLE***, ***GL\_TRUE***);

window = *glfwCreateWindow*(width, height, "Flappy", ***NULL***, ***NULL***);

if (window == ***NULL***) {

System.***err***.println("Could not create GLFW window!");

return;

}

GLFWVidMode vidmode = *glfwGetVideoMode*(*glfwGetPrimaryMonitor*());

*glfwSetWindowPos*(window, (vidmode.width() - width) / 2, (vidmode.height() - height) / 2);

*glfwSetKeyCallback*(window, new Input());

*glfwMakeContextCurrent*(window);

*glfwShowWindow*(window);

GL.*createCapabilities*();

*glEnable*(***GL\_DEPTH\_TEST***);

*glActiveTexture*(***GL\_TEXTURE1***);

*glEnable*(***GL\_BLEND***);

*glBlendFunc*(***GL\_SRC\_ALPHA***, ***GL\_ONE\_MINUS\_SRC\_ALPHA***);

System.***out***.println("OpenGL: " + *glGetString*(***GL\_VERSION***));

Shader.*loadAll*();

Matrix4f pr\_matrix = Matrix4f.*orthographic*(-10.0f, 10.0f, -10.0f \* 9.0f / 16.0f, 10.0f \* 9.0f / 16.0f, -1.0f, 1.0f);

Shader.*BG*.setUniformMat4f("pr\_matrix", pr\_matrix);

Shader.*BG*.setUniform1i("tex", 1);

Shader.*BIRD*.setUniformMat4f("pr\_matrix", pr\_matrix);

Shader.*BIRD*.setUniform1i("tex", 1);

Shader.*PIPE*.setUniformMat4f("pr\_matrix", pr\_matrix);

Shader.*PIPE*.setUniform1i("tex", 1);

level = new Level();

}

public void run() {

init();

long lastTime = System.*nanoTime*();

double delta = 0.0;

double ns = 1000000000.0 / 60.0;

long timer = System.*currentTimeMillis*();

int updates = 0;

int frames = 0;

while (running) {

long now = System.*nanoTime*();

delta += (now - lastTime) / ns;

lastTime = now;

if (delta >= 1.0) {

update();

updates++;

delta--;

}

render();

frames++;

if (System.*currentTimeMillis*() - timer > 1000) {

timer += 1000;

// System.out.println(updates + " ups, " + frames + " fps");

updates = 0;

frames = 0;

}

if (*glfwWindowShouldClose*(window))

running = false;

// glfwPollEvents();

}

*glfwDestroyWindow*(window);

*glfwTerminate*();

}

private void update() {

*glfwPollEvents*();

level.update();

if (level.isGameOver()) {

level = new Level();

}

}

private void render() {

*glClear*(***GL\_COLOR\_BUFFER\_BIT*** | ***GL\_DEPTH\_BUFFER\_BIT***);

level.render();

int error = *glGetError*();

if (error != ***GL\_NO\_ERROR***)

System.***out***.println(error);

*glfwSwapBuffers*(window);

}

public static void main(String[] args) {

new Main().start();

}

}

package com.project.flappy.graphics;

import static org.lwjgl.opengl.GL20.\*;

import java.util.HashMap;

import java.util.Map;

import com.project.flappy.maths.Matrix4f;

import com.project.flappy.maths.Vector3f;

import com.project.flappy.utils.ShaderUtils;

public class Shader {

public static final int ***VERTEX\_ATTRIB*** = 0;

public static final int ***TCOORD\_ATTRIB*** = 1;

public static Shader *BG*, *BIRD*, *PIPE*, *FADE*;

private boolean enabled = false;

private final int ID;

private Map<String, Integer> locationCache = new HashMap<String, Integer>();

public Shader(String vertex, String fragment) {

ID = ShaderUtils.*load*(vertex, fragment);

}

public static void loadAll() {

*BG* = new Shader("shaders/bg.vert", "shaders/bg.frag");

*BIRD* = new Shader("shaders/bird.vert", "shaders/bird.frag");

*PIPE* = new Shader("shaders/pipe.vert", "shaders/pipe.frag");

*FADE* = new Shader("shaders/fade.vert", "shaders/fade.frag");

}

public int getUniform(String name) {

if (locationCache.containsKey(name))

return locationCache.get(name);

int result = *glGetUniformLocation*(ID, name);

if (result == -1) {

System.***err***.println("Could not find uniform variable '" + name + "'!");

}

else {

locationCache.put(name, result);

}

return result;

}

public void setUniform1i(String name, int value) {

if (!enabled) enable();

*glUniform1i*(getUniform(name), value);

}

public void setUniform1f(String name, float value) {

if (!enabled) enable();

*glUniform1f*(getUniform(name), value);

}

public void setUniform2f(String name, float x, float y) {

if (!enabled) enable();

*glUniform2f*(getUniform(name), x, y);

}

public void setUniform3f(String name, Vector3f vector) {

if (!enabled) enable();

*glUniform3f*(getUniform(name), vector.x, vector.y, vector.z);

}

public void setUniformMat4f(String name, Matrix4f matrix) {

if (!enabled) enable();

*glUniformMatrix4fv*(getUniform(name), false, matrix.toFloatBuffer());

}

public void enable() {

*glUseProgram*(ID);

enabled = true;

}

public void disable() {

*glUseProgram*(0);

enabled = false;

}

}

package com.project.flappy.graphics;

import java.awt.image.BufferedImage;

import java.io.FileInputStream;

import java.io.IOException;

import javax.imageio.ImageIO;

import com.project.flappy.utils.BufferUtils;

import static org.lwjgl.opengl.GL11.\*;

public class Texture {

private int width, height;

private int texture;

public Texture(String path) {

texture = load(path);

}

private int load(String path) {

int[] pixels = null;

try {

BufferedImage image = ImageIO.*read*(new FileInputStream(path));

width = image.getWidth();

height = image.getHeight();

pixels = new int[width \* height];

image.getRGB(0, 0, width, height, pixels, 0, width);

} catch (IOException e) {

e.printStackTrace();

}

int[] data = new int[width \* height];

for (int i = 0; i < width \* height; i++) {

int a = (pixels[i] & 0xff000000) >> 24;

int r = (pixels[i] & 0xff0000) >> 16;

int g = (pixels[i] & 0xff00) >> 8;

int b = (pixels[i] & 0xff);

data[i] = a << 24 | b << 16 | g << 8 | r;

}

int result = *glGenTextures*();

*glBindTexture*(***GL\_TEXTURE\_2D***, result);

*glTexParameteri*(***GL\_TEXTURE\_2D***, ***GL\_TEXTURE\_MIN\_FILTER***, ***GL\_NEAREST***);

*glTexParameteri*(***GL\_TEXTURE\_2D***, ***GL\_TEXTURE\_MAG\_FILTER***, ***GL\_NEAREST***);

*glTexImage2D*(***GL\_TEXTURE\_2D***, 0, ***GL\_RGBA***, width, height, 0, ***GL\_RGBA***, ***GL\_UNSIGNED\_BYTE***, BufferUtils.*createIntBuffer*(data));

*glBindTexture*(***GL\_TEXTURE\_2D***, 0);

return result;

}

public void bind() {

*glBindTexture*(***GL\_TEXTURE\_2D***, texture);

}

public void unbind() {

*glBindTexture*(***GL\_TEXTURE\_2D***, 0);

}

}

package com.project.flappy.graphics;

import static org.lwjgl.opengl.GL11.\*;

import static org.lwjgl.opengl.GL15.\*;

import static org.lwjgl.opengl.GL20.\*;

import static org.lwjgl.opengl.GL30.\*;

import com.project.flappy.utils.BufferUtils;

public class VertexArray {

private int vao, vbo, ibo, tbo;

private int count;

public VertexArray(int count) {

this.count = count;

vao = *glGenVertexArrays*();

}

public VertexArray(float[] vertices, byte[] indices, float[] textureCoordinates) {

count = indices.length;

vao = *glGenVertexArrays*();

*glBindVertexArray*(vao);

vbo = *glGenBuffers*();

*glBindBuffer*(***GL\_ARRAY\_BUFFER***, vbo);

*glBufferData*(***GL\_ARRAY\_BUFFER***, BufferUtils.*createFloatBuffer*(vertices), ***GL\_STATIC\_DRAW***);

*glVertexAttribPointer*(Shader.***VERTEX\_ATTRIB***, 3, ***GL\_FLOAT***, false, 0, 0);

*glEnableVertexAttribArray*(Shader.***VERTEX\_ATTRIB***);

tbo = *glGenBuffers*();

*glBindBuffer*(***GL\_ARRAY\_BUFFER***, tbo);

*glBufferData*(***GL\_ARRAY\_BUFFER***, BufferUtils.*createFloatBuffer*(textureCoordinates), ***GL\_STATIC\_DRAW***);

*glVertexAttribPointer*(Shader.***TCOORD\_ATTRIB***, 2, ***GL\_FLOAT***, false, 0, 0);

*glEnableVertexAttribArray*(Shader.***TCOORD\_ATTRIB***);

ibo = *glGenBuffers*();

*glBindBuffer*(***GL\_ELEMENT\_ARRAY\_BUFFER***, ibo);

*glBufferData*(***GL\_ELEMENT\_ARRAY\_BUFFER***, BufferUtils.*createByteBuffer*(indices), ***GL\_STATIC\_DRAW***);

*glBindBuffer*(***GL\_ELEMENT\_ARRAY\_BUFFER***, 0);

*glBindBuffer*(***GL\_ARRAY\_BUFFER***, 0);

*glBindVertexArray*(0);

}

public void bind() {

*glBindVertexArray*(vao);

if (ibo > 0)

*glBindBuffer*(***GL\_ELEMENT\_ARRAY\_BUFFER***, ibo);

}

public void unbind() {

if (ibo > 0)

*glBindBuffer*(***GL\_ELEMENT\_ARRAY\_BUFFER***, 0);

*glBindVertexArray*(0);

}

public void draw() {

if (ibo > 0)

*glDrawElements*(***GL\_TRIANGLES***, count, ***GL\_UNSIGNED\_BYTE***, 0);

else

*glDrawArrays*(***GL\_TRIANGLES***, 0, count);

}

public void render() {

bind();

draw();

}

}

package com.project.flappy.input;

import org.lwjgl.glfw.GLFW;

import org.lwjgl.glfw.GLFWKeyCallback;

public class Input extends GLFWKeyCallback {

public static boolean[] *keys* = new boolean[65536];

public void invoke(long window, int key, int scancode, int action, int mods) {

*keys*[key] = action != GLFW.***GLFW\_RELEASE***;

}

public static boolean isKeyDown(int keycode) {

return *keys*[keycode];

}

}

package com.project.flappy.level;

import static org.lwjgl.glfw.GLFW.\*;

import com.project.flappy.graphics.Shader;

import com.project.flappy.graphics.Texture;

import com.project.flappy.graphics.VertexArray;

import com.project.flappy.input.Input;

import com.project.flappy.maths.Matrix4f;

import com.project.flappy.maths.Vector3f;

public class Bird {

private float SIZE = 1.0f;

private VertexArray mesh;

private Texture texture;

private Vector3f position = new Vector3f();

private float rot;

private float delta = 0.0f;

public Bird() {

float[] vertices = new float[] {

-SIZE / 2.0f, -SIZE / 2.0f, 0.2f,

-SIZE / 2.0f, SIZE / 2.0f, 0.2f,

SIZE / 2.0f, SIZE / 2.0f, 0.2f,

SIZE / 2.0f, -SIZE / 2.0f, 0.2f

};

byte[] indices = new byte[] {

0, 1, 2,

2, 3, 0

};

float[] tcs = new float[] {

0, 1,

0, 0,

1, 0,

1, 1

};

mesh = new VertexArray(vertices, indices, tcs);

texture = new Texture("res/bird.png");

}

public void update() {

position.y -= delta;

if (Input.*isKeyDown*(***GLFW\_KEY\_SPACE***))

delta = -0.13f;

else

delta += 0.01f;

rot = -delta \* 90.0f;

}

public void fall() {

delta = -0.15f;

}

public void render() {

Shader.*BIRD*.enable();

Shader.*BIRD*.setUniformMat4f("ml\_matrix", Matrix4f.*translate*(position).multiply(Matrix4f.*rotate*(rot)));

texture.bind();

mesh.render();

Shader.*BIRD*.disable();

}

public float getY() {

return position.y;

}

public float getSize() {

return SIZE;

}

}

package com.project.flappy.level;

import java.util.Random;

import com.project.flappy.graphics.Shader;

import com.project.flappy.graphics.Texture;

import com.project.flappy.graphics.VertexArray;

import com.project.flappy.input.Input;

import com.project.flappy.maths.Matrix4f;

import com.project.flappy.maths.Vector3f;

import static org.lwjgl.glfw.GLFW.\*;

public class Level {

private VertexArray background, fade;

private Texture bgTexture;

private int xScroll = 0;

private int map = 0;

private int score = 0;

private Bird bird;

private Pipe[] pipes = new Pipe[5 \* 2];

private int index = 0;

private float OFFSET = 5.0f;

private boolean control = true, reset = false;

private int lastCrossedPipeIndex = -1;

private Random random = new Random();

private float time = 0.0f;

public Level() {

float[] vertices = new float[] {

-10.0f, -10.0f \* 9.0f / 16.0f, 0.0f,

-10.0f, 10.0f \* 9.0f / 16.0f, 0.0f,

0.0f, 10.0f \* 9.0f / 16.0f, 0.0f,

0.0f, -10.0f \* 9.0f / 16.0f, 0.0f

};

byte[] indices = new byte[] {

0, 1, 2,

2, 3, 0

};

float[] tcs = new float[] {

0, 1,

0, 0,

1, 0,

1, 1

};

fade = new VertexArray(6);

background = new VertexArray(vertices, indices, tcs);

bgTexture = new Texture("res/bg.jpeg");

bird = new Bird();

createPipes();

}

private void createPipes() {

Pipe.*create*();

for (int i = 0; i < 5 \* 2; i += 2) {

pipes[i] = new Pipe(OFFSET + index \* 3.0f, random.nextFloat() \* 4.0f);

pipes[i + 1] = new Pipe(pipes[i].getX(), pipes[i].getY() - 12.5f);

index += 2;

}

score++;

}

private void updatePipes() {

int newIndex = 0;

pipes[index % 10] = new Pipe(OFFSET + index \* 3.0f, random.nextFloat() \* 4.0f);

pipes[(index + 1) % 10] = new Pipe(pipes[index % 10].getX(), pipes[index % 10].getY() - 12.5f);

if (lastCrossedPipeIndex != newIndex && -xScroll > pipes[newIndex].getX() + Pipe.*getWidth*() / 2) {

score++;

lastCrossedPipeIndex = newIndex; // Update the last crossed pipe index

System.***out***.println("Score: " + score); // Print the score

}

index += 2;

}

public void getScore() {

System.***out***.print(score);

}

public void update() {

if (control) {

xScroll--;

if (-xScroll % 335 == 0) map++;

if (-xScroll > 250 && -xScroll % 120 == 0)

updatePipes();

}

bird.update();

if (control && collision()) {

bird.fall();

control = false;

}

if (!control && Input.*isKeyDown*(***GLFW\_KEY\_SPACE***))

reset = true;

time += 0.01f;

}

private void renderPipes() {

Shader.*PIPE*.enable();

Shader.*PIPE*.setUniform2f("bird", 0, bird.getY());

Shader.*PIPE*.setUniformMat4f("vw\_matrix", Matrix4f.*translate*(new Vector3f(xScroll \* 0.05f, 0.0f, 0.0f)));

Pipe.*getTexture*().bind();

Pipe.*getMesh*().bind();

for (int i = 0; i < 5 \* 2; i++) {

Shader.*PIPE*.setUniformMat4f("ml\_matrix", pipes[i].getModelMatrix());

Shader.*PIPE*.setUniform1i("top", i % 2 == 0 ? 1 : 0);

Pipe.*getMesh*().draw();

}

Pipe.*getMesh*().unbind();

Pipe.*getTexture*().unbind();

}

private boolean collision() {

for (int i = 0; i < 5 \* 2; i++) {

float bx = -xScroll \* 0.05f;

float by = bird.getY();

float px = pipes[i].getX();

float py = pipes[i].getY();

float bx0 = bx - bird.getSize() / 2.0f;

float bx1 = bx + bird.getSize() / 2.0f;

float by0 = by - bird.getSize() / 2.0f;

float by1 = by + bird.getSize() / 2.0f;

float px0 = px;

float px1 = px + Pipe.*getWidth*();

float py0 = py;

float py1 = py + Pipe.*getHeight*();

if (bx1 > px0 && bx0 < px1) {

if (by1 > py0 && by0 < py1) {

return true;

}

}

}

return false;

}

public boolean isGameOver() {

return reset;

}

public void render() {

bgTexture.bind();

Shader.*BG*.enable();

Shader.*BG*.setUniform2f("bird", 0, bird.getY());

background.bind();

for (int i = map; i < map + 4; i++) {

Shader.*BG*.setUniformMat4f("vw\_matrix", Matrix4f.*translate*(new Vector3f(i \* 10 + xScroll \* 0.03f, 0.0f, 0.0f)));

background.draw();

}

Shader.*BG*.disable();

bgTexture.unbind();

renderPipes();

bird.render();

Shader.*FADE*.enable();

Shader.*FADE*.setUniform1f("time", time);

fade.render();

Shader.*FADE*.disable();

}

}

package com.project.flappy.level;

import com.project.flappy.graphics.Texture;

import com.project.flappy.graphics.VertexArray;

import com.project.flappy.maths.Matrix4f;

import com.project.flappy.maths.Vector3f;

public class Pipe {

private Vector3f position = new Vector3f();

private Matrix4f ml\_matrix;

private static float *width* = 1.5f, *height* = 8.0f;

private static Texture *texture*;

private static VertexArray *mesh*;

public static void create() {

float[] vertices = new float[] {

0.0f, 0.0f, 0.1f,

0.0f, *height*, 0.1f,

*width*, *height*, 0.1f,

*width*, 0.0f, 0.1f

};

byte[] indices = new byte[] {

0, 1, 2,

2, 3, 0

};

float[] tcs = new float[] {

0, 1,

0, 0,

1, 0,

1, 1

};

*mesh* = new VertexArray(vertices, indices, tcs);

*texture* = new Texture("res/pipe.png");

}

public Pipe(float x, float y) {

position.x = x;

position.y = y;

ml\_matrix = Matrix4f.*translate*(position);

}

public float getX() {

return position.x;

}

public float getY() {

return position.y;

}

public Matrix4f getModelMatrix() {

return ml\_matrix;

}

public static VertexArray getMesh() {

return *mesh*;

}

public static Texture getTexture() {

return *texture*;

}

public static float getWidth() {

return *width*;

}

public static float getHeight() {

return *height*;

}

}

package com.project.flappy.maths;

import java.nio.FloatBuffer;

import com.project.flappy.utils.BufferUtils;

public class Matrix4f {

public static final int ***SIZE*** = 4 \* 4;

public float[] elements = new float[***SIZE***];

public Matrix4f() {

}

public static Matrix4f identity() {

Matrix4f result = new Matrix4f();

for (int i = 0; i < ***SIZE***; i++) {

result.elements[i] = 0.0f;

}

result.elements[0 + 0 \* 4] = 1.0f;

result.elements[1 + 1 \* 4] = 1.0f;

result.elements[2 + 2 \* 4] = 1.0f;

result.elements[3 + 3 \* 4] = 1.0f;

return result;

}

public static Matrix4f orthographic(float left, float right, float bottom, float top, float near, float far) {

Matrix4f result = *identity*();

result.elements[0 + 0 \* 4] = 2.0f / (right - left);

result.elements[1 + 1 \* 4] = 2.0f / (top - bottom);

result.elements[2 + 2 \* 4] = 2.0f / (near - far);

result.elements[0 + 3 \* 4] = (left + right) / (left - right);

result.elements[1 + 3 \* 4] = (bottom + top) / (bottom - top);

result.elements[2 + 3 \* 4] = (far + near) / (far - near);

return result;

}

public static Matrix4f translate(Vector3f vector) {

Matrix4f result = *identity*();

result.elements[0 + 3 \* 4] = vector.x;

result.elements[1 + 3 \* 4] = vector.y;

result.elements[2 + 3 \* 4] = vector.z;

return result;

}

public static Matrix4f rotate(float angle) {

Matrix4f result = *identity*();

float r = (float) Math.*toRadians*(angle);

float cos = (float) Math.*cos*(r);

float sin = (float) Math.*sin*(r);

result.elements[0 + 0 \* 4] = cos;

result.elements[1 + 0 \* 4] = sin;

result.elements[0 + 1 \* 4] = -sin;

result.elements[1 + 1 \* 4] = cos;

return result;

}

public Matrix4f multiply(Matrix4f matrix) {

Matrix4f result = new Matrix4f();

for (int y = 0; y < 4; y++) {

for (int x = 0; x < 4; x++) {

float sum = 0.0f;

for (int e = 0; e < 4; e++) {

sum += this.elements[x + e \* 4] \* matrix.elements[e + y \* 4];

}

result.elements[x + y \* 4] = sum;

}

}

return result;

}

public FloatBuffer toFloatBuffer() {

return BufferUtils.*createFloatBuffer*(elements);

}

}

package com.project.flappy.maths;

public class Vector3f {

public float x, y, z;

public Vector3f() {

x = 0.0f;

y = 0.0f;

z = 0.0f;

}

public Vector3f(float x, float y, float z) {

this.x = x;

this.y = y;

this.z = z;

}

}

package com.project.flappy.utils;

import java.nio.ByteBuffer;

import java.nio.ByteOrder;

import java.nio.FloatBuffer;

import java.nio.IntBuffer;

public class BufferUtils {

private BufferUtils() {

}

public static ByteBuffer createByteBuffer(byte[] array) {

ByteBuffer result = ByteBuffer.*allocateDirect*(array.length).order(ByteOrder.*nativeOrder*());

result.put(array).flip();

return result;

}

public static FloatBuffer createFloatBuffer(float[] array) {

FloatBuffer result = ByteBuffer.*allocateDirect*(array.length << 2).order(ByteOrder.*nativeOrder*()).asFloatBuffer();

result.put(array).flip();

return result;

}

public static IntBuffer createIntBuffer(int[] array) {

IntBuffer result = ByteBuffer.*allocateDirect*(array.length << 2).order(ByteOrder.*nativeOrder*()).asIntBuffer();

result.put(array).flip();

return result;

}

}

package com.project.flappy.utils;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class FileUtils {

private FileUtils() {

}

public static String loadAsString(String file) {

StringBuilder result = new StringBuilder();

try {

BufferedReader reader = new BufferedReader(new FileReader(file));

String buffer = "";

while ((buffer = reader.readLine()) != null) {

result.append(buffer + '\n');

}

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

return result.toString();

}

}

package com.project.flappy.utils;

import static org.lwjgl.opengl.GL11.\*;

import static org.lwjgl.opengl.GL20.\*;

public class ShaderUtils {

private ShaderUtils() {

}

public static int load(String vertPath, String fragPath) {

String vert = FileUtils.*loadAsString*(vertPath);

String frag = FileUtils.*loadAsString*(fragPath);

return *create*(vert, frag);

}

public static int create(String vert, String frag) {

int program = *glCreateProgram*();

int vertID = *glCreateShader*(***GL\_VERTEX\_SHADER***);

int fragID = *glCreateShader*(***GL\_FRAGMENT\_SHADER***);

*glShaderSource*(vertID, vert);

*glShaderSource*(fragID, frag);

*glCompileShader*(vertID);

if (*glGetShaderi*(vertID, ***GL\_COMPILE\_STATUS***) == ***GL\_FALSE***) {

System.***err***.println("Failed to compile vertex shader!");

System.***err***.println(*glGetShaderInfoLog*(vertID));

return -1;

}

*glCompileShader*(fragID);

if (*glGetShaderi*(fragID, ***GL\_COMPILE\_STATUS***) == ***GL\_FALSE***) {

System.***err***.println("Failed to compile fragment shader!");

System.***err***.println(*glGetShaderInfoLog*(fragID));

return -1;

}

*glAttachShader*(program, vertID);

*glAttachShader*(program, fragID);

*glLinkProgram*(program);

*glValidateProgram*(program);

*glDeleteShader*(vertID);

*glDeleteShader*(fragID);

return program;

}

}

-----------------------------------------------------------------------Shaders----------------------------------------------------------------------------------

#version 330 core

layout (location = 0) out vec4 color;

in DATA

{

vec2 tc;

vec3 position;

} fs\_in;

uniform vec2 bird;

uniform sampler2D tex;

void main()

{

color = texture(tex, fs\_in.tc);

color \*= 2.0 / (length(bird - fs\_in.position.xy) + 2.5) + 0.6;

}

#version 330 core

layout (location = 0) in vec4 position;

layout (location = 1) in vec2 tc;

uniform mat4 pr\_matrix;

uniform mat4 vw\_matrix;

out DATA

{

vec2 tc;

vec3 position;

} vs\_out;

void main()

{

gl\_Position = pr\_matrix \* vw\_matrix \* position;

vs\_out.tc = tc;

vs\_out.position = vec3(vw\_matrix \* position);

}

#version 330 core

layout (location = 0) out vec4 color;

in DATA

{

vec2 tc;

} fs\_in;

uniform sampler2D tex;

void main()

{

color = texture(tex, fs\_in.tc);

if (color.w < 1.0)

discard;

}

#version 330 core

layout (location = 0) in vec4 position;

layout (location = 1) in vec2 tc;

uniform mat4 pr\_matrix;

uniform mat4 vw\_matrix = mat4(1.0);

uniform mat4 ml\_matrix = mat4(1.0);

out DATA

{

vec2 tc;

} vs\_out;

void main()

{

gl\_Position = pr\_matrix \* vw\_matrix \* ml\_matrix \* position;

vs\_out.tc = tc;

}

#version 330 core

layout (location = 0) out vec4 color;

uniform float time;

void main()

{

if (time > 1.0)

discard;

color = vec4(1.0, 1.0, 1.0, 1.0 - time);

}

#version 330 core

void main()

{

const vec4 vertices[6] = vec4[6]

(vec4( 1.0, -1.0, -0.5, 1.0),

vec4(-1.0, -1.0, -0.5, 1.0),

vec4( 1.0, 1.0, -0.5, 1.0),

vec4( 1.0, 1.0, -0.5, 1.0),

vec4(-1.0, -1.0, -0.5, 1.0),

vec4(-1.0, 1.0, -0.5, 1.0));

gl\_Position = vertices[gl\_VertexID];

}

#version 330 core

layout (location = 0) out vec4 color;

in DATA

{

vec2 tc;

vec3 position;

} fs\_in;

uniform vec2 bird;

uniform sampler2D tex;

uniform int top;

void main()

{

vec2 myTc = vec2(fs\_in.tc.x, fs\_in.tc.y);

if (top == 1)

{

myTc.y = 1 - myTc.y;

}

color = texture(tex, myTc);

if (color.w < 1.0)

{

discard;

}

color \*= 2.0 / (length(bird - fs\_in.position.xy) + 1.5) + 0.6;

color.w = 1.0;

}

#version 330 core

layout (location = 0) in vec4 position;

layout (location = 1) in vec2 tc;

uniform mat4 pr\_matrix;

uniform mat4 vw\_matrix = mat4(1.0);

uniform mat4 ml\_matrix = mat4(1.0);

out DATA

{

vec2 tc;

vec3 position;

} vs\_out;

void main()

{

gl\_Position = pr\_matrix \* vw\_matrix \* ml\_matrix \* position;

vs\_out.tc = tc;

vs\_out.position = vec3(vw\_matrix \* ml\_matrix \* position);

}

**Code explanation:**

Java code for a Flappy Bird game implemented using LWJGL (Lightweight Java Game Library) and OpenGL. This code contains several classes for handling graphics, input, shaders, and the game logic itself.

Here is a brief overview of the key classes:

**Main**: This class serves as the entry point for the game. It initializes GLFW (the windowing system), sets up OpenGL, and manages the game loop.

**Shader**: This class handles loading, compiling, and managing shaders used in the game. It provides methods to set various types of uniforms in the shaders.

**Texture**: This class loads and manages textures used in the game.

**VertexArray**: This class manages vertex array objects (VAOs) and vertex buffer objects (VBOs) for rendering.

**Input**: This class extends GLFWKeyCallback to handle keyboard input.

**Bird**: This class represents the bird character in the game. It handles its position, movement, and rendering.

**Level**: This class manages the game world, including background, pipes, and the bird. It handles updates and rendering.

**Pipe**: This class represents the pipes in the game. It handles their position, collision detection, and rendering.

**Matrix4f and Vector3f**: These classes are used for 3D transformations and vectors in the game.

**FileUtils and BufferUtils**: These utility classes handle file I/O and buffer operations, respectively.

**Shaders (in separate files):** These are GLSL shader programs for rendering different elements of the game, including the background, bird, pipes, and fade effect.

**Game Screenshots:**

|  |  |
| --- | --- |
| D:\FlappyBird\Project\FlappyBird\res\bg.jpeg | D:\FlappyBird\Project\FlappyBird\res\pipe.png  D:\FlappyBird\Project\FlappyBird\res\bird.png |

*Characters and enviroment*

|  |
| --- |
|  |

*Gameplay*

**RESULTS AND DISCUSSION:**

"Flappy Bird" had a profound impact on the gaming industry and indie game development. Its success demonstrated that a simple, free-to-play game with minimalistic graphics could achieve extraordinary popularity. Many developers were inspired to create their own versions or similar games, leading to a surge in the popularity of "Flappy Bird" clones.

Ultimately, "Flappy Bird" remains a notable example of how a straightforward game concept, combined with engaging gameplay and a touch of challenge, can lead to viral success in the mobile gaming world. Despite its relatively short lifespan in the app stores, its influence on the gaming industry and its status as a cultural phenomenon continue to be felt years after its release.

**CONCLUSION AND SCOPE OF FURTHER WORK:**

"Flappy Bird" stands as a remarkable example of the impact a simple yet engaging mobile game can have on the gaming industry. Its sudden rise to fame showcased the potential of indie developers to create highly popular titles without the need for extensive resources or complex graphics. The game's addictive nature and challenging gameplay kept players hooked, leading to widespread recognition and countless clones and imitations.

Future Scope:

While "Flappy Bird" itself may no longer be available for download, its legacy lives on in the form of numerous similar games and its influence on the design of mobile games. The success of "Flappy Bird" demonstrated the enduring appeal of games that prioritize intuitive controls, accessibility, and a degree of challenge. Game developers can draw inspiration from its straightforward mechanics to create new titles that captivate audiences.

Additionally, the "Flappy Bird" phenomenon highlights the potential for indie developers to make a significant impact in the gaming industry. With the continued growth of mobile gaming platforms and the availability of powerful game development tools, there are ample opportunities for creators to craft innovative and engaging experiences that capture the imagination of players worldwide.

In summary, "Flappy Bird" will be remembered as a pioneering mobile game that left an indelible mark on the gaming landscape. Its influence can be seen in the continued popularity of similar games and serves as a testament to the potential for innovative and accessible game design in the ever-evolving world of mobile gaming.

**BIBLIOGRAPHY:**

|  |  |
| --- | --- |
| ***1.*** | *https://www.lwjgl.org/guide* |
| ***2.*** | *https://www.opengl.org/sdk/libs/* |
| **3.** | https://www.w3schools.com/java/ |