“TypeCraft” Typing Speed System

Aidan Matthew Ong[1],David Brian Dimapilis [2] and Rudd Anthony 3 Of LBYC PEI-EQ5

[1] *De La Salle University - Manila*

[2] *Gokongwei College of Engineering*

1. INTRODUCTION

TypeCraft is a groundbreaking typing program designed to enhance the way students acquire typing skills, aligning with the Sustainable Development Goal (SDG) 4 of ensuring quality education for all. Through a blend of gamified learning, adaptive feedback, and simplified user interface, it aims to bolster digital literacy, individualize learning, and integrate seamlessly into existing educational frameworks, contributing to the global commitment to inclusive and equitable quality education. Unlike existing applications such as Typing.com, TypingClub, and KeyBlaze, which primarily focus on providing typing lessons and tracking progress, TypeCraft goes a step further. TypeCraft provides a dynamic typing journey, adapting to each learner's skill level and offering personalized feedback to improve accuracy and speed. Its user-friendly user interface allows educators to align typing exercises with curriculum needs, making it a comprehensive tool for boosting computer literacy and supporting the broader educational goals outlined in SDG 4. Moreover, TypeCraft's engaging and gamified interface motivates students to continue practicing and progressing. This level of engagement is a distinctive aspect of TypeCraft, as it transforms the often monotonous task of learning to type into an exciting and rewarding experience, reflecting the principles of quality education emphasized in SDG 4.

2. RELATED WORK

The program that the researchers made is a simple typing speed program. The development of typing speed programs has a rich history, with various applications designed to enhance typing skills and speed. One of the most notable in this field is Mavis Beacon Teaches Typing, a software application that has been a staple in typing education for decades. Mavis Beacon Teaches Typing was first released in 1987 and has since become one of the leading typing software programs. It offers a series of typing lessons, tests, and games designed to improve typing speed and accuracy. The program's user-friendly interface and structured lessons have made it a popular choice among educators and individuals seeking to enhance their typing skills.While Mavis Beacon set a standard in typing education, new programs like TypeCraft have built upon this foundation, introducing innovative features and methodologies. Unlike Mavis Beacon, which focuses primarily on structured lessons and practice tests, TypeCraft incorporates gamified learning, adaptive feedback, and customization aligned with curriculum needs.

3. PROGRAM FEATURES AND SCREENSHOTS

Feature 1: Displaying the Menu

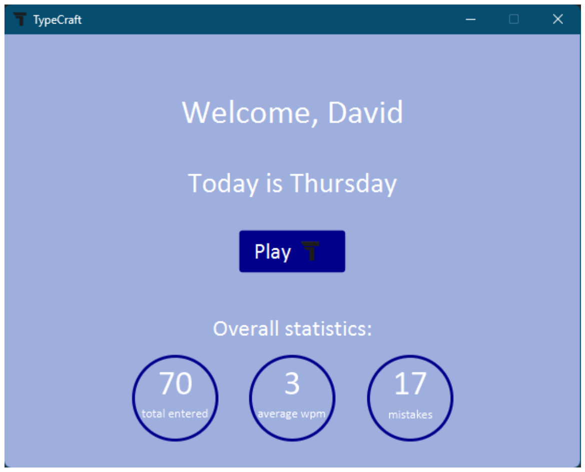


Figure 1 - Sample Output of the first feature (Displaying Menu).

In the first feature, The Java code provided represents a JavaFX application that orchestrates the display and functionality of the main menu. This application is structured to provide an engaging user interface, with the main menu being the central point of interaction. At the heart of the application is the primary stage, which serves as the main window. The initialization of this stage takes place in the start method, where several key attributes are defined. First, the window is set to be non-resizable, ensuring a consistent appearance across different systems. Next, the title "TypeCraft" is assigned to the window, and an application icon is set, adding a professional touch to the application's visual presentation. The main menu itself is defined in an external file named sample.fxml. This file, although not provided in the code snippet, would contain the layout and components of the main menu, such as buttons, labels, and other controls. The FXMLLoader class is utilized to load this FXML file, creating a Parent object that serves as the root of the scene.

Once the main menu's layout is loaded, a new scene is created with a fixed size of 600x450 pixels. This scene is then set as the primary stage's scene, and the window is displayed to the user using the show method. The fixed size ensures that the design and layout remain consistent, providing a uniform user experience. Beyond the main menu, the code includes a method for changing scenes, named changeScene. This method allows for seamless navigation between different parts of the application. By accepting an FXML file name as a parameter, it can load different layouts and set them as the new root of the scene. This functionality enables the application to transition from the main menu to other screens, enhancing the interactivity and user engagement.

Feature 2: Overall Statistics Function

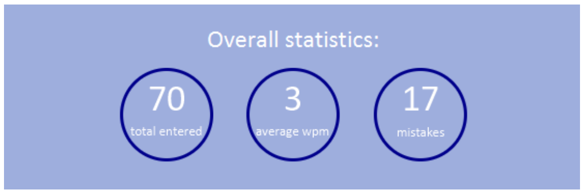


Figure 2 - Sample Output of the second feature

The second feature focuses on statistical analysis of numerical data across multiple files within a specified directory. The method sumUpNumbers scans through each file, reading three numbers from each line and summing them up, while also counting the total number of files processed. This results in an array containing the total counts of the three numbers and the total file count. Any reading errors are gracefully ignored, allowing the process to continue with other files. This functionality serves as a robust solution for gathering statistical insights from structured numerical data across a collection of files.

Feature 3: Timer and Statistics Calculator

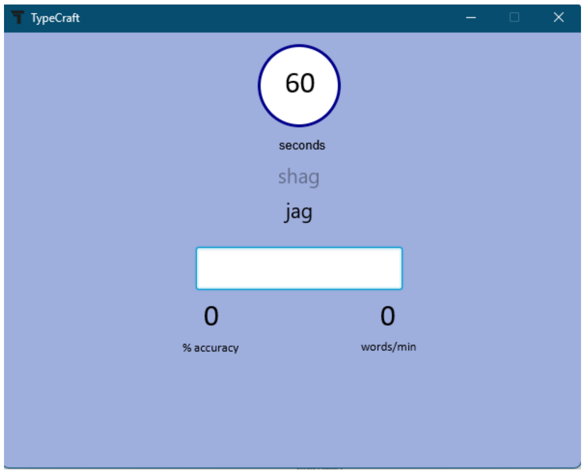


Figure 3 - Sample Output of the third feature

The third feature is the core of a typing game implemented using JavaFX. The class is responsible for managing the game's timer and calculating statistics related to the player's performance. Here's a brief explanation of each feature:

Timer Feature

The timer feature is implemented using a ScheduledExecutorService named executor, which schedules a Runnable task named r to execute every second. The task decrements a timer variable from 60 to -4, updating a Text object seconds to display the remaining time to the user. When the timer reaches -1, the game ends, and the user input is disabled. When it reaches -4, a "Play Again" button becomes visible and enabled, and the executor is shut down. This timer feature adds a sense of urgency to the game and controls the game's flow, providing a time-bound challenge to the player.

Statistics Calculator

The statistics calculator is responsible for tracking and displaying the player's performance during the game. It maintains two counters, countAll and counter, to track the total number of words attempted and the number of correct words typed, respectively. The method startGame is called on each key press and is responsible for starting the game, checking the user's input against the expected word, and updating the statistics. If the input is correct, a correct feedback animation is displayed using a thread running the fadeCorrect task. If incorrect, a wrong feedback animation is displayed using the fadeWrong task. The words per minute (wordsPerMin) and accuracy (accuracy) are calculated and displayed to the user in real-time. Additionally, the statistics are written to a file when the game ends, providing a record of the player's performance.

Feature 4: Startup Name Customization

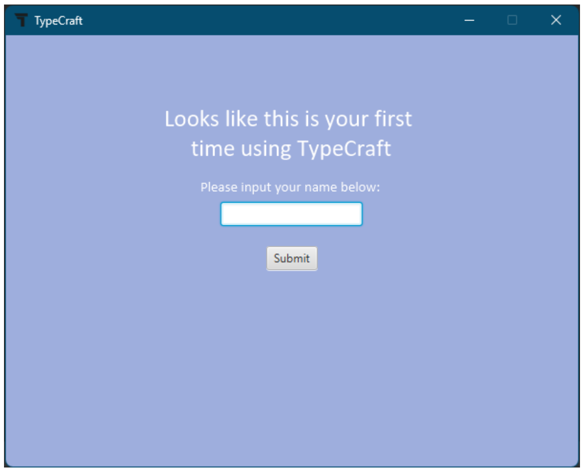
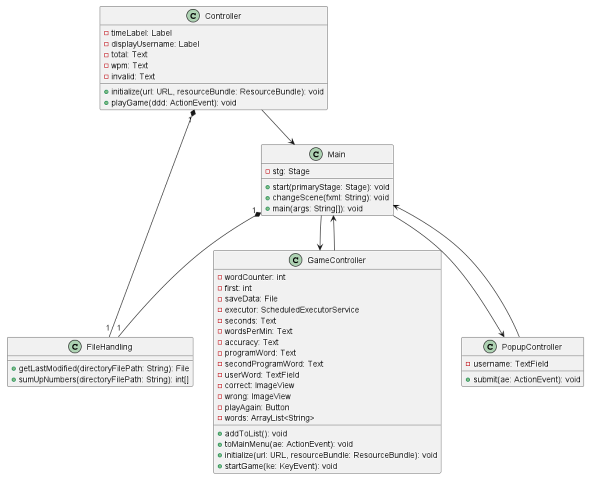

TypeCraft
Looks like this is your first time using TypeCraft
Please input your name below:
Submit

Figure 4 - Sample Output of the fourth feature

The fourth feature is utilizing The PopupController class which manages a popup feature that allows users to enter a username. When the "Submit" button is clicked, the submit method is called, retrieving the entered username from a TextField and writing it to a file named "username.txt" using a FileWriter. After writing the username, the method closes the FileWriter and changes the scene back to the main menu using the Main class's changeScene method. This popup feature provides a simple and effective way to capture user input and store it for later use, seamlessly integrating with the application's navigation flow.

4. UML DIAGRAM:


Controller
timeLabel: Label
displayUsername: Label
total: Text
wpm: Text
invalid: Text
initialize(url: URL, resourceBundle: ResourceBundle): void
playGame(ddd: ActionEvent): void
Main
stg: Stage
start(primaryStage: Stage): void
changeScene(fxml: String): void
main(args: String[]): void
GameController
wordCounter: int
first: int
saveData: File
executor: ScheduledExecutorService
seconds: Text
wordsPerMin: Text
accuracy: Text
programWord: Text
secondProgramWord: Text
userWord: TextField
correct: ImageView
wrong: ImageView
playAgain: Button
words: ArrayList<String>
addToList(): void
toMainMenu(ae: ActionEvent): void
initialize(url: URL, resourceBundle: ResourceBundle): void
startGame(ke: KeyEvent): void
FileHandling
getLastModified(directoryFilePath: String): File
sumUpNumbers(directoryFilePath: String): int[]
PopupController
username: TextField
submit(ae: ActionEvent): void
1
1
1
1

Click here for full size UML Diagram

This PlantUML Class Diagram represents the structure and relationships between five classes: Main, Controller, FileHandling, GameController, and PopupController. In this essay, we'll focus on how the four pillars of Object-Oriented Programming (OOP) - Encapsulation, Inheritance, Polymorphism, and Abstraction - are implemented in this diagram.

1. Encapsulation
Encapsulation is evident in the way the classes are structured. Each class has private attributes (denoted by the minus sign -) that are hidden from other classes, and public methods (denoted by the plus sign +) that provide controlled access to those attributes. For example, the Main class encapsulates the Stage object stg, and the GameController class encapsulates various attributes related to the game's state and UI components.

2. Inheritance
The diagram does not explicitly show any inheritance relationships between the classes, so it's not possible to directly identify the implementation of inheritance from the given information. Inheritance would typically be represented by an arrow with a hollow triangle head pointing from a subclass to a superclass, but no such relationships are depicted here.

3. Polymorphism
Polymorphism is not explicitly depicted in the diagram, but it can be inferred from the design. For example, the initialize method in both Controller and GameController classes may be an implementation of an interface method, allowing different behaviors in each class. Polymorphism allows objects of different classes to be treated as objects of a common superclass or interface, enabling more flexible and reusable code.

4. Abstraction
Abstraction is represented by the way the classes are designed to perform specific tasks, hiding the complexity from other parts of the system. For instance, the FileHandling class abstracts the details of file operations, providing methods like getLastModified and sumUpNumbers. The Main class interacts with FileHandling without needing to know the underlying implementation details.

Relationships
The diagram also shows various relationships between the classes:

Main has associations with GameController, PopupController, and FileHandling, indicating that it interacts with these classes.
Controller has associations with Main and FileHandling, reflecting its role in controlling the application's flow and interacting with file operations.
Both GameController and PopupController have associations with Main, signifying their connection to the main application logic.

5. CONCLUSION

In conclusion, TypeCraft's interactive and personalized approach to improving typing skills directly contributes to SDG 4 (Quality Education) by providing a versatile and accessible tool for learners of all ages to enhance their abilities in an engaging and motivating manner. The program's focus on continuous improvement and individualized learning experiences align with the global goal of promoting inclusive and equitable quality education for all, contributing to a more skilled and knowledgeable global population.

5. REFERENCES

Helping students boost typing skills. A union of professionals. (n.d.). https://www.uft.org/news/teaching/linking-learning/helping-students-boost-typing-skills

Gedeon, T. (n.d.). How technology can help drive SDG 4: Quality Education. LinkedIn. https://www.linkedin.com/pulse/how-technology-can-help-drive-sdg-4-quality-education-talal-gedeon

The benefits of keyboarding, the key to your future. Typesy. (n.d.).

https://www.typesy.com/the-benefits-of-keyboarding-the-key-to-your-future/

The best typing and keyboarding sites for classrooms. Common Sense Education. (n.d.). https://www.commonsense.org/education/best-in-class/the-best-typing-and-keyboarding-sites-for-classrooms#:~:text=Learning%20to%20type%20with%20accuracy,what%20they%20want%20to%20say.

2011-2023, (c) Copyright skillsyouneed.com. (n.d.). Stop pecking at your keyboard: The many benefits of learning how to type fast and accurately. SkillsYouNeed. https://www.skillsyouneed.com/rhubarb/benefits-of-typing.html

Top 10 best typing software [typing tutors for 2023]. Software Testing Help. (2023, June 28). https://www.softwaretestinghelp.com/best-typing-software/

Typing skills: Why they should be taught to early learners. AWE Learning. (2019, January 3). https://awelearning.com/blog/typing-skills-why-they-should-be-taught-to-early-learners/

admin, P. by. (2021, February 16). Web-based typing programs for Schools. Typing Agent. https://blog.typingagent.com/web-based-typing-programs-for-schools/

Why invest in typing programs for Schools. Touch-type Read and Spell (TTRS). (2023, May 30). https://www.readandspell.com/typing-programs-for-schools

10, S., 17, J., &amp; 15, A. (2022, June 20). Why typing is important for 21st Century learners. Learning Without Tears. https://www.lwtears.com/blog/why-typing-important-21st-century-learners

6. APPENDIX (PROGRAM CODES)

```
Main.java

package sample;



import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.image.Image;
import javafx.stage.Stage;


import java.io.IOException;


public class Main extends Application {


// A reference to the primary stage, used for changing scenes

private static Stage stg;


// The main entry point of the JavaFX application

@Override

public void start(Stage primaryStage) throws Exception {

stg = primaryStage;

primaryStage.setResizable(false);


// Load the FXML file 'sample.fxml' and create a scene with it

Parent root = FXMLLoader.load(getClass().getResource("sample.fxml"));


primaryStage.setTitle("TypeCraft"); // Set the title for the application
window

primaryStage.getIcons().add(new  Image("images/icon.png"));  //  Set  the
application icon

primaryStage.setScene(new Scene(root, 600, 450)); // Set the scene with a
fixed width and height

primaryStage.show(); // Display the application window

}


// Method to switch scenes based on the provided FXML file name

public void changeScene(String fxml) throws IOException {
```

```
// Load the FXML file specified by the 'fxml' parameter and create a new
scene

Parent pane = FXMLLoader.load(getClass().getResource(fxml));



// Set the new scene as the root scene of the primary stage

stg.getScene().setRoot(pane);

}



// The main method that launches the JavaFX application

public static void main(String[] args) {

launch(args);

}

}



Controller.java

package sample;



import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Label;

import javafx.scene.text.Text;



import java.io.File;

import java.io.FileNotFoundException;

import java.io.IOException;

import java.net.URL;

import java.text.DateFormat;
```

```java
import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Locale;

import java.util.ResourceBundle;

import java.util.Scanner;


public class Controller implements Initializable {


// FXML elements representing UI components

@FXML

private Label timeLabel;

@FXML

private Label displayUsername;

@FXML

private Text total;

@FXML

private Text wpm;

@FXML

private Text invalid;


// This method is automatically called when the FXML file is loaded.

// It initializes the controller and sets up the initial state of the UI.

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {


// Check if the 'username.txt' file is not empty (contains user data)

File newFile = new File("username.txt");
if (newFile.length() != 0) {

Scanner reader = null;

try {

reader = new Scanner(newFile);

} catch (FileNotFoundException e) {

e.printStackTrace();

}

String data = reader.nextLine();


// Display a personalized welcome message with the user's name

displayUsername.setText("Welcome, " + data);

}


// Set the current day of the week

Date date = new Date();

Locale locale = new Locale("en");

DateFormat formatter = new SimpleDateFormat("EEEE", locale);

String strDay = formatter.format(date);

timeLabel.setText("Today is " + strDay);


// Fetch and display data for test statistics

int[] data = FileHandling.sumUpNumbers("src/data");

total.setText(String.valueOf(data[0])); // Total typing attempts

wpm.setText(String.valueOf(Math.round(data[1]*1.0/data[3]))); // Words Per Minute (WPM)

invalid.setText(String.valueOf(data[2])); // Count of invalid attempts

}
```

```
// This method is called when the user clicks the 'Play Game' button.

// It handles the logic to switch between different scenes (views) based
on user data.

public void playGame(ActionEvent ddd) throws IOException {

Main m = new Main();



// Check if the 'username.txt' file is empty (no user data)

File newFile = new File("username.txt");

if (newFile.length() == 0) {

try {

// If no user data, switch to 'popup.fxml' for username input

m.changeScene("popup.fxml");

} catch (IOException e) {

e.printStackTrace();

}

} else {

// If user data is available, switch to 'game.fxml' to start the typing
test

m.changeScene("game.fxml");

}

}

}



FileHandling.java

package sample;



import java.io.BufferedReader;

import java.io.File;
import java.io.FileReader;

import java.io.IOException;


public class FileHandling {


/**

* Gets the most recently modified file in the specified directory.

*

* @param directoryFilePath The path of the directory to search for files.

* @return The most recently modified file in the directory, or null if the
directory is empty.

*/

public static File getLastModified(String directoryFilePath) {

File directory = new File(directoryFilePath);

File[] files = directory.listFiles(File::isFile);

long lastModifiedTime = Long.MIN_VALUE;

File chosenFile = null;


if (files != null) {

// Loop through the files in the directory

for (File file : files) {

// Check if the current file's last modified time is greater than the
current maximum

if (file.lastModified() > lastModifiedTime) {

// Update the chosenFile and lastModifiedTime to the current file's values

chosenFile = file;

lastModifiedTime = file.lastModified();

}
```

```
}

}



return chosenFile;

}



/**

* Sums up the numbers in the files present in the specified directory and
calculates some statistics.

*

* @param directoryFilePath The path of the directory containing the files.

* @return An array of integers containing the following statistics:

* [0] Total count of the first number in each file

* [1] Total count of the second number in each file

* [2] Total count of the third number in each file

* [3] Total count of files in the directory

*/

public static int[] sumUpNumbers(String directoryFilePath) {

File directory = new File(directoryFilePath);

File[] files = directory.listFiles(File::isFile);

int[] returnArray = new int[4];

String[] arr;

int counter = 0;

if (files != null) {

// Loop through the files in the directory

for (File file : files) {

counter++; // Increment the counter for each file processed

try {
```

```
// Read the content of the current file using BufferedReader

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

String st;

while ((st = br.readLine()) != null) {

// Split each line by ";" and convert the values to integers for summing
up

arr = st.split(";");

returnArray[0] += Integer.parseInt(arr[0]);

returnArray[1] += Integer.parseInt(arr[1]);

returnArray[2] += Integer.parseInt(arr[2]);

}

br.close(); // Close the BufferedReader to release resources


} catch (IOException ignored) {

// Ignore any IOException and continue processing other files

}

}

}

returnArray[3] = counter; // Store the total count of files in the
directory in the last element

return returnArray;

}

}


GameController.java

package sample;


import javafx.event.ActionEvent;
```

```java
import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.scene.image.ImageView;

import javafx.scene.input.KeyCode;

import javafx.scene.input.KeyEvent;

import javafx.scene.text.Text;


import java.io.*;

import java.net.URL;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Date;

import java.util.ResourceBundle;

import java.util.concurrent.Executors;

import java.util.concurrent.ScheduledExecutorService;

import java.util.concurrent.TimeUnit;


public class GameController implements Initializable {


private int wordCounter = 0;

private int first = 1;


private File saveData;
```

```java
ScheduledExecutorService executor = Executors.newScheduledThreadPool(1);


@FXML
public Text seconds;
@FXML
private Text wordsPerMin;
@FXML
private Text accuracy;
@FXML
private Text programWord;
@FXML
private Text secondProgramWord;

@FXML
private TextField userWord;

@FXML
private ImageView correct;
@FXML
private ImageView wrong;

@FXML
private Button playAgain;

ArrayList<String> words = new ArrayList<>();

// add words to array list
```

```java
public void addToList() {
BufferedReader reader;
try {
// Open the wordsList file for reading
reader = new BufferedReader(new FileReader("wordsList"));
String line = reader.readLine();
while (line != null) {
words.add(line);
// read next line
line = reader.readLine();
}
reader.close();
} catch (IOException e) {
e.printStackTrace();
}
}


// Navigate back to the main menu when the button is clicked
public void toMainMenu(ActionEvent ae) throws IOException {
Main m = new Main();
m.changeScene("sample.fxml");
}


@Override
public void initialize(URL url, ResourceBundle resourceBundle) {


playAgain.setVisible(false);
playAgain.setDisable(true);

seconds.setText("60");

addToList();

Collections.shuffle(words);

programWord.setText(words.get(wordCounter));

secondProgramWord.setText(words.get(wordCounter+1));

wordCounter++;


Date date = new Date();

SimpleDateFormat formatter = new SimpleDateFormat("dd-MM-yyyy HH-mm-ss");

saveData = new File("src/data/"+formatter.format(date).strip()+".txt");


try {

if (saveData.createNewFile()) {

System.out.println("File created: " + saveData.getName());

} else {

System.out.println("File already exists.");

}

} catch (IOException e) {

e.printStackTrace();

}



}



private int timer = 60;



Runnable r = new Runnable() {
```

```
@Override

public void run() {

if (timer > -1) {

seconds.setText(String.valueOf(timer));

timer -= 1;

}


else {

if (timer == -1) {

userWord.setDisable(true);

userWord.setText("Game over");


try {

// Write the test statistics to the saveData file

FileWriter myWriter = new FileWriter(saveData);

myWriter.write(countAll +";");

myWriter.write(counter +";");

myWriter.write(String.valueOf(countAll-counter));

myWriter.close();

} catch (IOException e) {

e.printStackTrace();

}

}


if (timer == -4) {

playAgain.setVisible(true);

playAgain.setDisable(false);
executor.shutdown();

}


timer -= 1;

}

}

};


Runnable fadeCorrect = new Runnable() {

@Override

public void run() {

correct.setOpacity(0);

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

correct.setOpacity(50);

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

correct.setOpacity(100);

try {

Thread.sleep(200);

} catch (InterruptedException e) {
```

```
e.printStackTrace();

}

correct.setOpacity(0);


}

};


Runnable fadeWrong = new Runnable() {

@Override

public void run() {

wrong.setOpacity(0);

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

wrong.setOpacity(50);

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

wrong.setOpacity(100);

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();
```

```
}
wrong.setOpacity(0);
}
};

private int countAll = 0;
private int counter = 0;

// Start the typing game and calculate the statistics based on user input
public void startGame(KeyEvent ke) {
// Only gets called once
if (first == 1) {
first = 0;
// Start the timer using the ScheduledExecutorService
executor.scheduleAtFixedRate(r, 0, 1, TimeUnit.SECONDS);
}

if (ke.getCode().equals(KeyCode.ENTER)) {
String s = userWord.getText();
String real = programWord.getText();
countAll++;

// Check if the user's input is correct
if (s.equals(real)) {
counter++;
wordsPerMin.setText(String.valueOf(counter));
```

```
// Start a thread to display the correct feedback animation

Thread t = new Thread(fadeCorrect);

t.start();


} else {

// Start a thread to display the wrong feedback animation

Thread t = new Thread(fadeWrong);

t.start();

}

userWord.setText("");

// Calculate and update the accuracy based on the current statistics

accuracy.setText(String.valueOf(Math.round((counter  *  1.0  /  countAll)  *
100)));

programWord.setText(words.get(wordCounter));

secondProgramWord.setText(words.get(wordCounter+1));

wordCounter++;

}

}

}

PopupController.java

package sample;


import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.scene.control.TextField;


import java.io.FileWriter;

import java.io.IOException;
```

```
public class PopupController {


@FXML

private TextField username;


// This method is called when the "Submit" button is clicked in the popup
window

public void submit(ActionEvent ae) throws IOException {

// Get the entered username from the text field

String name = username.getText();


// Create a FileWriter object to write the username to the "username.txt"
file

FileWriter myObj = new FileWriter("username.txt");


// Write the username to the file

myObj.write(name);


// Close the FileWriter to release resources

myObj.close();


// Create a new Main object to change the scene back to the main menu

Main m = new Main();

m.changeScene("sample.fxml");

}



}
```

```
TypeCraft.puml
@startuml


class Main {
-stg: Stage
+start(primaryStage: Stage): void
+changeScene(fxml: String): void
+main(args: String[]): void
}


class Controller {
-timeLabel: Label
-displayUsername: Label
-total: Text
-wpm: Text
-invalid: Text
+initialize(url: URL, resourceBundle: ResourceBundle): void
+playGame(ddd: ActionEvent): void
}


class FileHandling {
+getLastModified(directoryFilePath: String): File
+sumUpNumbers(directoryFilePath: String): int[]
}


class GameController {
-wordCounter: int
-first: int

-saveData: File

-executor: ScheduledExecutorService

-seconds: Text

-wordsPerMin: Text

-accuracy: Text

-programWord: Text

-secondProgramWord: Text

-userWord: TextField

-correct: ImageView

-wrong: ImageView

-playAgain: Button

-words: ArrayList<String>

+addToList(): void

+toMainMenu(ae: ActionEvent): void

+initialize(url: URL, resourceBundle: ResourceBundle): void

+startGame(ke: KeyEvent): void

}


class PopupController {

-username: TextField

+submit(ae: ActionEvent): void

}


Main --> GameController

Main --> PopupController

Main "1" *-- "1" FileHandling
```

```
Controller --> Main

Controller "1" *-- "1" FileHandling

GameController --> Main

PopupController --> Main


@enduml
```

7. INDIVIDUAL CONTRIBUTIONS

David Brian Dimapilis

- Did majority of the program code
- Made the video presentation
- Made the JAVAFX interface
- Made the UML Class Diagram

Aidan Matthew Ong

- Did the skeleton code of the program
- Made the poster

Rudd Anthony

- Made the powerpoint presentation
- Designed the java fx interface with diff fonts/colors

8. Video Presentation

Video Presentation can be found here:

https://youtu.be/_qQehHCPwxA