Touch Type Workout: the keyboard touch-type training system for seniors and kids.

Kyrylo Krysko.
University of Michigan.
coder@umich.edu

Adnan K Shaout.
University of Michigan.
shaout@umich.edu

Figure 1. The UI of Touch Type Workout. Main window with Lessons and test tab. Here user starts the typing lessons or test by clicking “Start Lesson” or “Start Test” buttons. The characters to type (input string) displayed over the nice keyboard image. When task is started, the keyboard shows the letter that needs to be pressed and results of user typing displayed in input string marking red the errors.

ABSTRACT

This report presents the development of Touch Type Workout (TTW), a standalone application designed to improve typing skills of kids and seniors. The project was inspired by the observation of the challenges faced by my parents and kids when interacting with computer keyboards. I found that despite the extreme age difference there are a lot of similarity in their performance, behavior and emotional response to the challenges.

TTW differentiates itself through a clean user-friendly interface and personalized feedback based on user performance. Typing without looking on keyboard is enabled by muscle memory, which is also sometimes called procedural memory, and is involved with learning and mastering motor skills. To gain this skill human typically needs to spend 10-15 hours on practicing. The situation becomes trickier when it comes to training kids and seniors since their behavior, motoric and motivation are quite different from adults (usual computer target audience).

Keywords: Touch Typing, Children, Senior, User Interface, HCI, Software Engineering, Python, Waterfall Model, Touch Typing Metrics.

1. INTRODUCTION

The Importance of Touch Typing in the Modern World. In the digital age, the ability to type quickly and accurately is a crucial skill. Touch typing, the ability to type without looking at the keyboard, is particularly important. It allows for increased productivity and efficiency, as individuals can focus on the screen and the task at hand, rather than searching for keys. This skill is essential in many professional contexts, including programming, data entry, writing, and communication. Moreover, as education increasingly incorporates digital tools, touch typing becomes a vital skill for students as well.

The Process of Gaining Touch Typing Skills. Learning to touch type involves memorizing the keyboard layout and training your fingers to find the keys automatically. The

process typically starts with learning the home row keys (the middle row of the keyboard), followed by the top and bottom rows. Over time, through repetition and practice, muscle memory develops, allowing the typist to find the keys without conscious thought. Touch typing also involves learning proper typing posture and finger placement. The typist should sit upright, with their feet flat on the floor and their elbows at a right angle. The fingers should rest lightly on the home row keys, with each finger responsible for certain keys.

Challenges of Touch Typing for Kids and Seniors. For children, the main challenges in learning to touch type often relate to motor skills and concentration. Young children may struggle with the fine motor control required to hit individual keys [1], and they may find it difficult to remember the keyboard layout. Additionally, children may have shorter attention spans, making sustained practice challenging. Seniors, on the other hand, may face physical challenges such as reduced dexterity or vision impairments [3], which can make touch typing more difficult. They may also struggle with the memorization aspect, particularly if they are not familiar with the QWERTY keyboard layout. Furthermore, seniors may face challenges in adapting to new technology, particularly if they have not used computers extensively in the past. This observations allows to assume that we can treat seniors and kids in the same way and address their need with a single approach relevant for both audiences.

Lack of Applications Training Touch Typing Skills for Seniors and Kids. While there are numerous touch typing training applications available, many are not designed with children or seniors in mind. Children's applications often focus more on entertainment than effective learning, while applications for adults may be too complex or fast-paced for seniors. There is a clear gap in the market for touch typing applications that are tailored to the specific needs and abilities of these demographics. This is where the Touch Type Workout (TTW) application comes in, providing a user-friendly and effective solution for kids and seniors to learn and practice touch typing.

Structure. This report outlines the process of developing TTW, from initial concept to final implementation. The report is structured into sections detailing inspiration and research, concept, and implementation. The Inspiration and research explains how I come up with the TTW application idea and what research has been made prior to the project launch. Concept shows the basic ideas of the application and differentiation from similar solutions. Implementation explains in depth the software development process applied to TTW development and provides architectural details about application.

2. INSPIRATION AND RESEARCH

The idea for Touch Type Workout was born out of personal observations and experiences. I am a parent to two young children, and I also have elderly parents who, despite primarily using touchscreen devices, occasionally need to use my desktop computer. Their experiences with keyboard were markedly different from my own, and yet strikingly similar to each other's.

My children, with their boundless curiosity and eagerness to explore, approached the keyboard with a sense of wonder. Their lack of familiarity with the letter placements led to slow, hunt-and-peck typing. They were eager to learn, but the standard keyboard layout seemed to be an overwhelming puzzle to them.

My parents, on the other hand, approached the keyboard with a sense of trepidation. Having not grown up with computers as an integral part of their daily lives, they were cautious, almost fearful, in their interactions. They worried about pressing the wrong keys or causing irreparable damage.

In both cases, I noticed a lack of confidence that was not present in my own interactions with the keyboard. My children and parents seemed to be intimidated by the keyboard, leading to emotional reactions - frustration from my children when they couldn't type as quickly as they wanted, and anxiety from my parents when faced with a seemingly insurmountable learning curve.

These observations led me to realize that there was a need for a touch typing solution specifically designed for kids and seniors. A solution that could turn the daunting task of learning to type into a manageable, and even enjoyable, activity. A solution that could build confidence and reduce the emotional stress associated with learning a new skill. And thus, the concept of Touch Type Workout was born.

A comprehensive review of existing literature and products was conducted to understand the current state of user interfaces for children and seniors, as well as touch typing metrics. This research revealed a gap in the market for touch typing solutions specifically designed for these demographics, with a focus on user-friendly interfaces and personalized feedback.

There are three predictors of high typing performance [2]:

- Unambiguous finger-to-key mapping, where a letter is consistently pressed by the same finger,
- Active preparation of upcoming keystrokes
- Minimal global hand motion.

Those findings should be considered during development of lessons and tests exercise. Challenge users at most in those predictors will help to develop most efficient training program.

3. CONCEPT

TTW is designed to fill the identified gap in the market. It differentiates itself from existing solutions through a clean, minimalistic, and easy-to-understand user interface. The application collects and records user performance data, providing personalized recommendations for improvement based on these statistics.

We can categorizes educational technology into different genres: Computer as Tutor, Computer as Tool, Computer as Tutee, and Computer-Supported Collaborative Learning (CSCL) [1]. Each genre represents a different way in which children can interact with and learn from technology. The CSCL model was chosen to design the application to get the most comfortable and user friendly outcome.

The training system should encourage consistent keystrokes [3]. This means that the time between each keystroke should be as uniform as possible. This could be achieved through exercises that promote rhythm and pacing in typing.

As a result, the concept of Touch Type Workout is rooted in two fundamental principles: **user-friendly design** and **personalized feedback.**

User-Friendly Design (Figure 1). The importance of a good and clear design cannot be overstated, especially when the target users are children and seniors. These demographics may not be as familiar or comfortable with technology as others, and thus, the design of the application plays a crucial role in their experience and understanding of it.

A well-designed application is intuitive, minimizing the learning curve for new users. It uses visual cues and a logical layout to guide users through the application, making it easy for them to find what they need. In the case of TTW, the design is minimalistic and clean, reducing visual clutter and focusing the user's attention on the essential elements. The aesthetic is pleasing but not distracting, creating a positive user experience.

The design also takes into account the specific needs of the target users. For children, this means using larger buttons and incorporating elements of play to make learning fun. For seniors, this means ensuring that text is readable and that the interface is not overwhelming.

Children can be involved in the design process of new technologies [1]. They can play different roles such as user, tester, informant, and design partner. I started to develop UI at a concept phase and involve my children in the process by choosing best options from keyboard design, fonts, colors, and later testing the application and getting a feedback.

Personalized Feedback. The core feature of TTW is its ability to record user performance and provide personalized feedback. This feature sets TTW apart from many existing touch typing applications, which often focus solely on teaching the keyboard layout.

A lower variance in the time between keystrokes indicates better temporal consistency and thus better typing skill [3]. The training system should encourage consistent keystrokes. This means that the time between each keystroke should be as uniform as possible. This could be achieved through exercises that promote rhythm in typing.

TTW records three key metrics: typing speed, accuracy, and rhythm. Speed measures how quickly the user can type, accuracy tracks the number of errors made, and rhythm assesses the consistency of the typing speed. By recording these metrics, TTW can provide users with a clear picture of their typing skills and areas for improvement.

The feedback provided by TTW is not generic; it is tailored to the individual user based on their performance data. This personalized feedback is more effective in helping users improve because it targets their specific weaknesses. For example, if a user types quickly but with many errors, TTW might suggest slowing down to improve accuracy. On the other hand, if a user types slowly but accurately, TTW might suggest exercises to increase speed.

Figure 2. The Statistics tab of the main window displays graphs of performance evolution within 3 key metrics, speed, accuracy and rhythm.

By combining a user-friendly design with personalized feedback, TTW offers a unique solution for teaching touch typing to kids and seniors. It not only teaches them the skill but also empowers them to improve and progress at their own pace.

4. IMPLEMENTATION

Here is the Touch Type Workout demo YouTube link: *https://youtu.be/bRKCiKmNqNg*

The full code listing available in APPENDIX A. CODE LISTING

Software Development Model.

The Waterfall model of software development was employed for this project. This traditional model includes sequential phases of concept development, requirements definition, design, coding, testing, and maintenance. Each phase was completed before the next was started, ensuring a systematic progression of the project. Concept, requirements and design phases results in documentation (Figure 3)

This model was chosen because it is most basic and straight-forward approach that suits student project perfectly.

Figure 3. Project documentation.

The first phase involved the creation of the concept document. This document served as the foundation for the project, outlining the main idea, purpose, and features of the application. Essentially, it served as a statement of the project's goals, providing a clear vision for what the application aimed to achieve.

Upon completion of the concept document, the project moved into the requirements phase. This phase focused on defining what the product must do in a precise and unambiguous manner. The resulting requirements document listed all the features of the product, as well as the environment in which it would operate and its intended usage. This document served as a roadmap for the development of the application, detailing the specific functionalities that needed to be implemented.

The design phase followed the requirements phase. During this phase, the requirements were translated into a detailed product architecture in the design document. This document provided all the necessary information to move into the programming phase, including the structure of the application, the technologies to be used, and the methods for implementing the required features.

In a typical Waterfall model, there is the possibility to revert back one step to refine the outputs of the previous phase. However, due to time constraints in this project, this iterative refinement process was not feasible. Despite this, the structured approach to documentation creation ensured a clear and organized development process, facilitating the successful implementation of the TTW application.

Architecture

TTW was developed using Python, with the PySide library for creating the user interface and the Matplotlib library for rendering statistics graphs. The user interface consists of a main window with two tabs: one for training and one for statistics. The training tab features a large keyboard, a list of lessons and tests, and a string for typing. The statistics tab displays graphs of speed, accuracy, and rhythm.

UI

The UI was implemented with QT library. All widgets were combined in QTDesigner application that allows built the necessary elements in user friendly way.

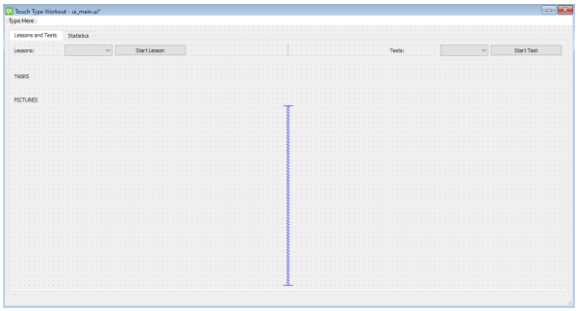

Figure 4. UI creation in **QTDesigner** application shipped with **PySide** Python library.

Synergy of PySide and Python allows building complex interfaces interactively and fast. Once UI file is done it is compiled to the Python format and redy to be imported into the main application.

The “TASKS” QLabel widget used to show user current assignment (text that needs to be typed). The “PICTURES” QLabel widget used do display keyboard images in main widget.

The UI is loading keyboard images dynamically according to user actions. The keyboard images shipped in JPEG format. They include blank keyboard and set of images for every key that user might press that highlight necessary key in blue color.

The custom font was developed from hand-written letters to satisfy pleasant and clear look and feel (Figure 5).

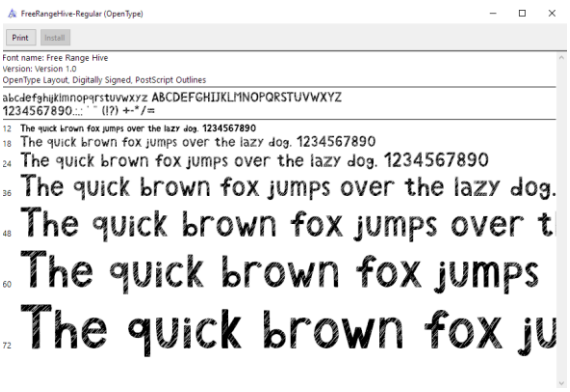

Figure 5. The TTW custom font created from hand-written images.

Application Data

TTW has two streams of data, input (lessons and tests) and outputs (user statistics). The lessons and tests are pre-defined and stored in JSON files. The user statistics recorded to JSON file constantly every time users using the application.

Application modules

Besides UI files and data storage items the TTW is a python file with three classes: **StatisticCanvas**, **MyStaticMplCanvas** and **TouchType**. The first two classes utilized to **matplot** library to display graph in the UI. The TouchType class is a hart of application that contains all application logic and functionality (Table 1).

Function Name	Inputs	Outputs	Description
`__init__`			Class constructor. Responsible for UI setup, UI function calls, provides attributes to store application data.
init_ui	Paths to JSON files with lessons, tests and statistics	Stores lessons and tests data in class attributes.	Loads lesson and tests data in the UI. Creates statistic file for recording user performance.
reset_ui	Lesson_started variable	Statistics data to ui widget	Reverts application UI to it's default state after lesson or test been performed. Displays statistic of last lesson or test. Calls 3 method to calculate performance: cps_to_wpm(), errors_rate(), rhythm()
start_lesson	lesson_name, lesson_data	Reset statistics attributes: time, wpm_lesson_time, wpm_lesson_characters, errors	Activated by "Start Lesson" button. Resets program flow logic. Calls start_sequence() function.
start_test	test_name, test_data	test_started trigger set to True	Activated by "Start Test" button. Resets program flow logic. Calls start_sequence() function.
reload_stat			Activated by "Reload Statistic" button. Calls recommendation() function
keyPressEvent	Key pressed event, active key	Sets program flow variables	Controls application lessons and test execution logic. It catches the event of keyboard hit and for each hit determines application behavior. 1) Checks if the pressed key was correct, 2) measures time for each user action along with errors and records the data, 3) Calls sequence_wpm(), record_statistics(), reset_ui() and start_sequence() functions.
check_user_input	task_string, user_string	colored_string	Compare task string with user input and display results in UI. Incorrect characters paint red
set_next_picture	task_string, current_string	keyboard_image_path	Displays key to press for user in UI
start_sequence	lesson_started, lesson_name, lesson_data	lesson_string	Display sequence of strings for current task in UI. Calculates number of characters in the string for statistics
recommendation	statistics_data	recommendation_string	Analyze user behavior based on recorded data and outputs recommendation for user in UI on how to improve the results.
rythm	key_stamps list	rhythm_value	Calculate rhythm, consistency of typing based on list of time stamps for each character of the task
errors_rate	task_string	user_string	Calculates number of incorrect keys pressed by user
cps_to_wpm	wpm_lesson_characters, wpm_lesson_time	wpm	Calculates Words Per Minute based on character per second data
sequence_wpm	wpm_lesson_time, current_time	sequence_time	Calculates total time of typing sequence by user
record_statistics	wpm_lesson_characters, wpm_lesson_time, errors		Records lesson or test statistics to a file

Table 1. **TouchType** class methods.

Recommendation System

The **core feature of TTW** is a Recommendation System that show user how he/she can improve touch-typing results based on statistics data of a last exercise. We tracking three metrics: typing speed, error rate and rhythm of typing.

Typing Speed (in Words Per Minute) = (characters per second * 60) / letters in word

Errors Rate: Percentage of errors = (number of errors/number of characters in the task)*100

Rhythm. The rhythm value provides a measure of the speed and consistency of typing. A lower rhythm value indicates faster and more consistent typing, since the time between subsequent key presses is less. A higher rhythm value indicates slower and/or less consistent typing, since the time between subsequent key presses is more.

We assign high or low value to each metric. The study [3] found that typing speeds ranged from 20 to 104 net words per minute among the typists examined. If words per minute less than 20 characters we set Typing Speed to “low”, otherwise it is “high”. If user made more than 50% of errors we set Errors Rate to “high”, otherwise it is “low”. A lower variance in the time between keystrokes indicates better temporal consistency and thus better typing skill [3]. We set Rhythm to “low” if it is above 5, otherwise it is “high”.

Next, we providing user a recommendation based on the current combination of low and high values for each metric (Table 2).

Metrics Combination	Recommendation
wpm_high and error_high and rhythm_high	Reduce speed and focus on accuracy. Try to maintain a steady rhythm when typing.
wpm_high and error_high and rhythm_low	Slow down your typing speed and concentrate on accuracy.
wpm_high and error_low and rhythm_high	Your rhythm is inconsistent. Try to establish a steady pace to increase efficiency.
wpm_high and error_low and rhythm_low	Your typing speed and accuracy are impressive, and you maintain a steady rhythm!
wpm_low and error_high and rhythm_high	Practice typing at a slower pace, concentrate on accuracy, try to develop a consistent rhythm.
wpm_low and error_high and rhythm_low	Focus more on accuracy and gradually increase your speed.
wpm_low and error_low and rhythm_high	You need to improve your typing speed and establish a steady rhythm.
wpm_low and error_low and rhythm_low	Accuracy and rhythm are great! Focus on increasing your typing speed through practice

Table 2. Recommendation table: metrics value combinations for providing feedback.

6. FUTURE IMPROVEMENTS

The application is bug free and fully functional, though I can see a potential in improving it in some areas. We can introduce support for multiple users, so many people can log in to their account on the same computer ant work independently. Currently application available on Windows OS only, so we can implement cross platform support.

We can also develop more advanced recommendation system by introducing more metrics and increase range from low and high to more gradual values. Additional metrics used in the study [2] to evaluate typing, such as cleaning and smoothing, offline labeling of markers, transformation to keyboard coordinate system, and identifying executing fingers, can be used to track progress and provide feedback to users in the training system.

When we create learning technology, we need to make sure it's easy to use (usability) and that it helps students learn (learning outcomes) [1]. We need to check how well it's working while we're creating it (formative evaluation) and after we've finished (summative evaluation). This process can take a long time and can be difficult, but it's important to make sure the technology is effective and beneficial for the students. Strategic improvements can relay on gathering and analyzing statistics from different users for a longer period of time.

5. CONCLISION

The development of TTW has resulted in a unique application that addresses the typing needs of kids and seniors. By focusing on a user-friendly interface and personalized feedback, TTW provides a tailored solution for these demographics. Future work could explore the integration of more advanced analytics and recommendation system to further enhance the user experience.

Need for the Application: The Touch Type Workout (TTW) application was developed in response to a clear gap in the market for touch typing applications that cater to the specific needs of children and seniors. The idea for the application was inspired by personal observations of the challenges faced by these demographics when interacting with computer keyboards.

Unique Features: TTW differentiates itself from existing solutions through a clean, user-friendly interface and personalized feedback based on user performance. The application records three key metrics: typing speed, accuracy, and rhythm, and provides personalized recommendations for improvement based on these statistics.

Implementation: The application was developed using the Waterfall model of software development, which includes sequential phases of concept development, requirements definition, design, coding, testing, and maintenance. TTW was developed using Python, with the PySide library for creating the user interface and the Matplotlib library for rendering statistics graphs.

Performance: The application appears to be performing well, with the report stating that it is bug-free and fully functional. The application's recommendation system, which provides feedback based on a user's typing speed, error rate, and rhythm, is a particularly notable feature.

Future Improvements: Despite its current success, there is potential for further improvement of the application. Future enhancements could include support for multiple users, cross-platform support, and improved statistical metrics and recommendation logic.

7. REFERENCES

[1] Amy Bruckman, Alisa Bandlow, and Andrea Forte. 2007. HCI for Kids. https://eva.fing.edu.uy/pluginfile.php/52761/mod_resource/content/0/biblioteca/Bruckman_Bandlow_-_HCI_for_Kids_-_IN_The_Human-Computer_Interaction_Handbook.pdf

[2] Anna Maria Feit Daryl Weir Antti Oulasvirta. 2016. How We Type: Movement Strategies and Performance in Everyday Typing. https://dl.acm.org/doi/pdf/10.1145/2858036.2858233

[3] Timothy A. Salthouse. 1984. Effects of Age and Skill in Typing. https://uva.theopenscholar.com/files/vcap/files/effects_of_age_and_skill_7.pdf

8. APPENDIX A. CODE LISTING.

```
"""
ECE579 Embedded Systems

Final Project - Touch Type Workout - The keyboard training system for kids and seniors
"""

import os
import time
import json
from PySide2 import QtWidgets, QtCore, QtGui
from ui import ui_main
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.figure import Figure


class StatisticCanvas(FigureCanvas):
    def __init__(self, parent=None):

        fig = Figure(dpi=100)
        self.axes = fig.subplots(3, 1, sharex=True)
        self.compute_initial_figure()

        FigureCanvas.__init__(self, fig)
        self.setParent(parent)

        FigureCanvas.setSizePolicy(self, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Expanding)
        FigureCanvas.updateGeometry(self)


class MyStaticMplCanvas(StatisticCanvas):
    def __init__(self, json_file, parent=None):
        self.json file = json file
        super().__init__(parent)

        self.statistics_data = None
        self.times = None
        self.wpm = None
        self.errors = None
        self.rhythm = None

    def load_data(self):

        with open(self.json_file) as f:
            self.statistics_data = json.load(f)

        self.times = list(self.statistics data.keys())

        # Calculate WPM
        self.wpm = []
        for value in self.statistics_data.values():
            cps = value['characters']/value['time']
            wpm = (cps * 60) / 5
            self.wpm.append(wpm)

        # Calculate errors
        self.errors = []
        for value in self.statistics data.values():
            error_value = int((value['errors']/value['characters'])*100)
            self.errors.append(error_value)

        # Get rhythm
        self.rhythm = []
        for value in self.statistics_data.values():
            self.rhythm.append(value['rhythm'])

    def plot(self):
        """
        Draw graphs from data
        """

        linewidth = 4

        self.axes[0].plot(self.times, self.wpm, 'r', label='Words Per Minute',
linewidth=linewidth)
        self.axes[1].plot(self.times, self.errors, 'b', label='Errors %',
linewidth=linewidth)
        self.axes[2].plot(self.times, self.rhythm, 'g', label='Typing Rhythm',
linewidth=linewidth)

    def compute initial figure(self):

        self.load_data()
        self.plot()

        for axe in self.axes:
            axe.legend()
            axe.tick_params(axis='both', which='major', labelsize=8)

        self.axes[0].figure.tight_layout()

    def update_figure(self):

        for axe in self.axes:
            axe.clear()

        self.load_data()
        self.plot()

        for axe in self.axes:
            axe.legend()
            axe.tick_params(axis='both', which='major', labelsize=8)

        self.axes[0].figure.tight layout()
        self.draw()


class TouchType(QtWidgets.QMainWindow, ui_main.Ui_TouchType):
    def __init__(self):
        super(TouchType, self).__init__()
        self.setupUi(self)
        font = QtGui.QFont('Free Range Hive')
        font.setPointSize(48)
        self.labTasks.setFont(font)
        font.setPointSize(32)
        self.labRecommendation.setFont(font)
        self.labTasks.setText('PRESS  <font color="red">START LESSON</font> OR <font
color="red">START TEST</font>')

        # Data
        self.statistic = f'{user}/Documents/touch_type_stat.json'
        self.keyboard_blank = f'{root}/data/images/_blank.png'
        self.keyboard_all = f'{root}/data/images/_all.png'
        self.lessons = f'{root}/data/lessons.json'
        self.tests = f'{root}/data/tests.json'
        self.lessons_data = None
        self.tests_data = None

        # Lesson flow control
        self.lesson_started = False
        self.lesson_name = None
        self.sequence_ended = False
        self.sequence_text = None
        self.number_of_sequences = 0
        self.current_sequence = 0
        self.current_string = 0
        self.string_length = None
        self.user_input = ''

        # Test flow control
        self.test_started = False
        self.test_name = None

        # Statistic flow control
        self.start_wpm = False
        self.time = None
        self.wpm_lesson_time = 0
        self.wpm_lesson_characters = 0
        self.errors = 0
        self.key_stamps = []

        # Setup UI with data
        self.init_ui()

        # Setup graph
        self.canvas = MyStaticMplCanvas(self.statistic, self)
        self.layStatistics.addWidget(self.canvas)

        # UI calls
        self.btnStartLesson.pressed.connect(self.start lesson)
        self.btnStartTest.pressed.connect(self.start_test)
        self.btnReloadStatistics.pressed.connect(self.reload_stat)

    # UI setup
    def reset_ui(self):

        pixmap = QtGui.QPixmap(self.keyboard_blank)
        self.labPictures.setPixmap(pixmap)
        self.labTasks.clear()

        if self.lesson started:
            string = 'LESSON'
        else:
            string = 'TEST'

        # Display lesson statistics
        self.labTasks.setText(f'{string} <font color="red">COMPLETE!</font> '
                              f'WPM: {self.cps_to_wpm()}, '
                              f'ERR: {self.errors_rate()}%, '
                              f'RHM: {self.rhythm()}')

    def init_ui(self):

        # Create empty statistic file if it is not exists:
        if not os.path.exists(self.statistic):
            with open(self.statistic, 'w') as file_content:
                json.dump({}, file_content, indent=4)

        # Tests and Lessons
        # Load keyboard
        pixmap = QtGui.QPixmap(self.keyboard_blank)
        self.labPictures.setPixmap(pixmap)

        # Load lessons and tests
        with open(self.lessons, 'r') as file_content:
            self.lessons data = json.load(file content)

        with open(self.tests, 'r') as file_content:
            self.tests_data = json.load(file_content)

        self.comLessons.addItems(self.lessons_data.keys())
        self.comTests.addItems(self.tests_data.keys())

        # Show recommendations
        self.recommendation()

    # Statistics
    def recommendation(self):
        """
        Read statistics of last session and provide recommendations
        """

        with open(self.statistic) as f:
            statistics_data = json.load(f)

        last_key = len(statistics_data.keys()) - 1
        last_session_data = statistics_data.get(str(last_key))

        if not last session data:
            print('>> The Statistics data is not available!')
            return

        # Get parameters data
        wpm = self.cps_to_wpm(last_session_data['characters'], last_session_data['time'])
        errors_rate = self.errors_rate(last_session_data['characters'],
last_session_data['errors'])
        rhythm = last_session_data["rhythm"]
        # print(f'Last Session data: {wpm}, {errors_rate}, {rhythm}')

        # Calibrate parameters
        wpm high = False
        wpm_low = False
        error high = False
        error_low = False
        rhythm_high = False
        rhythm_low = False

        if wpm < 20:
            wpm_low = True
        else:
            wpm_high = True
        if errors rate > 50:
            error_high = True
        else:
            error_low = True
        if rhythm > 5:
            rhythm_low = True
        else:
            rhythm_high = True

        # Calculate recommendation
        string = f'You are awesome! WPM: <font color="red">{wpm}</font>, ' \
                 f'ERRORS:  <font color="red">{errors_rate}</font>, ' \
                 f'RHYTHM: <font color="red">{rhythm}</font>!'

        # High WPM, High Error Rate, High Rhythm
        if wpm_high and error_high and rhythm_high:
            string = f'Reduce speed and focus on accuracy. Try to maintain a steady
rhythm when typing.'

        # High WPM, High Error Rate, Low Rhythm
        if wpm_high and error_high and rhythm_low:
            string = f'Slow down your typing speed and concentrate on accuracy.'

        # High WPM, Low Error Rate, High Rhythm
        if wpm_high and error_low and rhythm_high:
            string = f'Your rhythm is inconsistent. Try to establish a steady pace to
increase efficiency.'

        # High WPM, Low Error Rate, Low Rhythm
        if wpm_high and error_low and rhythm_low:
            string = f'Your typing speed and accuracy are impressive, and you maintain a
steady rhythm!'

        # Low WPM, High Error Rate, High Rhythm
        if wpm_low and error_high and rhythm_high:
```

```
            string = f'Practice typing at a slower pace, concentrate on accuracy, try to
develop a consistent rhythm.'

        # Low WPM, High Error Rate, Low Rhythm
        if wpm_low and error_high and rhythm_low:
            string = f'Focus more on accuracy and gradually increase your speed.'

        # Low WPM, Low Error Rate, High Rhythm
        if wpm_low and error_low and rhythm_high:
            string = f'You need to improve your typing speed and establish a steady
rhythm.'

        # Low WPM, Low Error Rate, Low Rhythm
        if wpm_low and error_low and rhythm_low:
            string = f'Accuracy and rhythm are great! Focus on increasing your typing
speed through practice.'

        self.labRecommendation.setText(string)

    def rhythm(self):
        """
        The rhythm value provides a measure of the speed and consistency of typing.

        A lower rhythm value indicates faster and more consistent typing,
        since the time between subsequent key presses is less.
        A higher rhythm value indicates slower and/or less consistent typing,
        since the time between subsequent key presses is more.
        """

        # Calculate the differences between subsequent timestamps
        intervals = [j - i for i, j in zip(self.key_stamps[:-1], self.key_stamps[1:])]

        # Get rhythm as average of intervals
        rhythm = sum(intervals) / len(intervals)

        # return round(rhythm, 2)
        return int(rhythm*10)

    def errors_rate(self, characters=None, errors=None):
        """
        Calculates number of incorrect keys pressed by user
        """

        if not characters:
            rate = self.errors / self.wpm_lesson_characters
        else:
            rate = errors / characters

        return int(rate * 100)

    def cps_to_wpm(self, characters=None, lesson_time=None):
        """
        words per minute = (characters per second * 60) / letters in word
        """

        if not characters:
            cps = self.wpm_lesson_characters/self.wpm_lesson_time
        else:
            cps = characters / lesson_time

        wpm = (cps*60)/5

        return int(wpm)

    def sequence_wpm(self):
        """
        Calculate sequence typing time (seconds)
        """

        sequence_time = time.time() - self.time
        self.wpm_lesson_time += sequence_time
        self.start_wpm = False

        # print(f'sequence_time = {sequence_time}')

    def record_statistics(self):
        """
        Calculate and record for each lesson or test session:
            - Words per Minute

        statistics = { 'session index': {session data}}
            session data = WPM: [letters, time]
        """

        # Load existing stat
        with open(self.statistic, 'r') as file_content:
            statistic_data = json.load(file_content)

        # Calculate Stat
        session_data = {'characters': self.wpm_lesson_characters,
                        'time': self.wpm_lesson_time,
                        'errors': self.errors,
                        'rhythm': self.rhythm()}

        if not statistic_data.keys():  # First launch
            statistic_data[0] = session_data
        else:
            number_of_sessions = len(statistic_data.keys())
            statistic_data[number_of_sessions] = session_data

        # Save updated stat
        with open(self.statistic, 'w') as file_content:
            json.dump(statistic_data, file_content, indent=4)

    # Flow control
    def check user input(self, user text):
        """
        Paint green correct keys, red - incorrect
        """

        colored_string = ""

        for correct_character, user_character in zip(self.sequence_text, user_text):
            # Correct characters
            if correct_character == user_character:
                colored string += f"<font color='green'>{user character}</font>"
            # Incorrect characters
            else:
                colored_string += f"<font color='red'>{user_character}</font>"

        # Preserve the rest of the original string after the user's input
        colored_string += self.sequence_text[len(user_text):]

        # Update the label text
        self.labTasks.setText(colored_string)

    def set_next_picture(self):

        # Get pressed key
        if self.sequence text[self.current string] == ' ':
            key = '_space'
        elif self.sequence_text[self.current_string] == '.':
            key = '_dot'
        elif self.sequence_text[self.current_string] == ',':
            key = '_comma'
        else:
            key = self.sequence_text[self.current_string].upper()

        # Show next letter in UI
        pixmap = f'{root}/data/images/{key}.png'
        self.labPictures.setPixmap(pixmap)

    def start_sequence(self):
        """
        Display sequence of strings for current lesson
        """

        if self.lesson_started:
            self.sequence_text =
self.lessons_data[self.lesson_name]['sequences'][self.current_sequence]
        else:
            self.sequence_text =
self.tests data[self.test name]['sequences'][self.current sequence]

        self.labTasks.setText(self.sequence_text)

        pixmap =
f'{root}/data/images/{self.sequence_text[self.current_string].upper()}.png'
        self.labPictures.setPixmap(pixmap)

        self.current_string = 1
        self.string_length = len(self.sequence_text)

        # Stat
        self.wpm lesson characters += self.string length

    def keyPressEvent(self, event):
        """
        Lesson Flow control
        """

        if self.lesson_started or self.test_started:

            # Check if it was a correct key
            task_letter = self.sequence_text[self.current_string - 1]
            pressed letter = chr(event.key())
            self.user_input += pressed_letter

            # Statistic
            # Words per minute
            if not self.start_wpm:
                # print(f'Start timing WPM')
                self.time = time.time()
                self.start_wpm = True

            # Key errors
            if task_letter != pressed_letter and not self.sequence_ended:
                self.errors += 1

            # Rhythm
            self.key_stamps.append(time.time())

            # Display errors in UI
            self.check_user_input(self.user_input)

            # Process all keys
            if self.current_string == self.string_length:
                if self.number_of_sequences == self.current_sequence + 1:
                    # End of lesson
                    # print('END SESSION')
                    # print('End timing WPM')
                    self.lesson_started = False
                    self.test_started = False
                    self.sequence_wpm()
                    self.record_statistics()
                    self.reset_ui()

                    return

                # End of Sequence
                pixmap = QtGui.QPixmap(self.keyboard_all)
                self.labPictures.setPixmap(pixmap)
                if self.sequence ended:
                    # Last letter of sequence
                    self.sequence_ended = False
                    self.user_input = ''
                    self.current_string = 0
                    self.current_sequence += 1
                    self.start_sequence()
                    return

                self.sequence_ended = True
                # print('End timing WPM')
                self.sequence wpm()
                return

            self.set_next_picture()

            # Update string counter
            self.current_string += 1

    # UI calls
    def start_lesson(self):

        self.lesson_name = self.comLessons.currentText()

        self.number_of_sequences = len(self.lessons_data[self.lesson_name]['sequences'])

        # Reset Flow
        self.user_input = ''
        self.current_sequence = 0
        self.current_string = 0
        self.lesson_started = True

        # Reset stat
        self.time = None
        self.wpm lesson time = 0
        self.wpm_lesson_characters = 0
        self.errors = 0
        del self.key_stamps[:]

        # Run sequence
        self.start_sequence()

        self.labPictures.setFocus()  # Allow application to catch SPACE key

    def start_test(self):

        self.test name = self.comTests.currentText()
        self.number_of_sequences = len(self.tests_data[self.test_name]['sequences'])
        self.user input = ''
        self.current_sequence = 0
        self.current_string = 0
        self.test_started = True
        self.start_sequence()

        self.labPictures.setFocus()  # Allow application to catch SPACE key

    def reload_stat(self):

        self.canvas.update_figure()
        self.recommendation()


if __name__ == "__main__":

    root = os.path.dirname(os.path.abspath(__file__))
    user = os.path.expanduser('~')
    app = QtWidgets.QApplication([])
    touch_type = TouchType()
    touch_type.setWindowIcon(QtGui.QIcon('{0}/icons/touch_type.ico'.format(root)))
    touch_type.show()
    app.exec_()
```