math_2022_043

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА

Факультет математики та інформатики
кафедра математичного моделювання

Розробка програмного забезпечення для догляду за станом серцево-судинної системи

Кваліфікаційна робота

Рівень вищої освіти - другий (магістерський)

Виконав:
студент 6 курсу, 607 групи
Спіжавка Станіслав Іванович
Керівник:
кандидат фізико-математичних наук,
доцент Івасюк Г. П.

До захисту допущено
на засіданні кафедри
протокол № 8 від 6 грудня 2022 р.
Зав. кафедрою _____ проф. Черевко І.М.

Чернівці – 2022

Анотація

Створено додаток за допомогою кросплатформного фреймворку для графічних програм Kyvi мови програмування Python. Додаток розроблений для людей, які хочуть слідкувати за станом серцево-судинної системи. Основна мета роботи спростити для користувача проходження тесту Руф'є та підвищити точність показників.

Ключові слова: програмне забезпечення, серцево-судинна система, кросплатформений додаток, класи та об'єкти.

Анотація

The application was created using the cross-platform Python framework for graphical applications Kyvi. This application is designed for people who want to monitor the state of the cardiovascular system. The main purpose of the application is to make it easier for the user to pass the Rufier test and improve the accuracy of the indicators.

Key words: software, cardiovascular system, cross-platform application, classes and objects.

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів наукових досліджень інших авторів мають посилання на відповідне джерело.

_____ С.І. Спіжавка
(підпис)

Зміст

Вступ

З розвитком технологій значно підвищився рівень життя людей, загалом покращився розвиток медицини та науки.

За допомогою гаджетів і програм, які до них додаються, можна полегшити собі життя, вдосконалити необхідні навички та відпочити.

На жаль, у світі існують люди, які не можуть жити повноцінним життям, так як в них є проблеми із серцево-судинною системою. Людина може жити і навіть не знати, що в неї не все в порядку зі здоров'ям, що їй потрібно звернутися до лікаря. Вирішити такого роду проблему можуть сучасні технології.

Цього року я провів доволі багато часу в лікарні. Туди я потрапив через проблеми зі здоров'ям. Проводячи дні там, я побачив що кожного ранку в коридорі збиралося декілька дітей і медсестра змушувала їх присідати і щось рахувати. Мені стало цікаво, що вони роблять. Підійшовши до неї і запитавши, що відбувається, я отримав відповідь, що ці діти проходять тест Руф'є.

Дивлячись на те як його проходили, я зрозумів що вони зіштовхуються з певними труднощами в тому що потрібно одночасно присідати, слідкувати за часом та рахувати пульс. Через це точність результатів була не найкраща. Тому мені спало на думку розробити додаток, який допоміг би людям проходити цей тест і покращив би точність отриманих даних.

Розробка здійснювалася за допомогою кросплатформного фреймворку Python для графічних програм Kyvi. Даний фреймворк зручний тим, що додаток можна буде скомпілювати під будь-яку операційну систему, не переписуючи код.

Робота складається зі вступу, трьох розділів, висновків, списку використаної літератури та додатків. У першому розділі описана постановка задачі, другий розділ присвячений опису використаних технологій для написання програми. Третій розділ є основним, у ньому описуються алгоритм розробки додатку та інтерфейс користувача.

Розділ. 1 Постановка задачі

Тест був розроблений французьким лікарем Джеймсом-Едвардом Руф’є (1875-1965) і запропонований доктором Діксоном у статті 1950 року як метод медичного та моторного контролю.

Ще в 2009 році в Україні Міністерство охорони здоров'я затвердило порядок проведення щорічних оглядів учнів шкільних та дошкільних закладів. Такі обстеження включають вимірювання стандартних параметрів учня: зріст, вага, артеріальний тиск. Крім того, необхідно здати лабораторні аналізи, відвідування спеціалістів, пробу Руф’є.

Сьогодні ви зіткнетеся з експертами, які стверджують, що тест Rufier давно застарів і тому досить глючний. У рідкісних випадках проба Руф'є має високе значення, але при цьому кардіолог і ЕКГ серця не знаходять відхилень. Проте наразі це єдиний тест, який ділить учнів на три групи з фізкультури: спеціальну, підготовчу та основну.

До основної групи увійшли діти, у яких не було виявлено жодних проблем зі здоров'ям. Ці школярі можуть проходити кроси, брати участь у змаганнях та брати участь в уроках загальної фізкультури.

До окремих груп належать школярі, стан здоров'я яких перешкоджає нормальній фізичній активності. Зазвичай такі діти страждають хронічними захворюваннями і перебувають на обліку в аптечній групі.

У деяких випадках учні об’єднуються в підготовчу групу. Це відбувається, коли дитина не має симптомів, але має незадовільний індекс Руф’є

Пробу Руф'є проводять у державній поліклініці за місцем прописки/проживання дітям старше 6 років. Це обов'язкова процедура перед початком навчального року.

Першим етапом тесту Руф’є є вимірювання частоти серцевих скорочень у спокої. Дитину кладуть на диван і дають відпочити кілька хвилин. Після цього вимірюють пульс і артеріальний тиск.

Другий етап – фізкультхвилинка, на якій дитина повинна зробити 30 присідань за 45 секунд. І тут багато лікарів і батьків помиляються - потрібно не просто сісти 30 разів, а сісти правильно. Лікарі та мами хочуть переконатися, що стопи дитини знаходяться на відстані 20 см одна від одної, щоб п’яти не відривалися від землі під час присідання, щоб спина була прямою, а руки витягнуті вперед або злегка зігнуті в ліктях (це дозволяє добре баланс).

В магазині мобільних додатків “Play Маркет” вже є розроблений подібний додаток (рис. 1).

Рис. 1. Мобільний додаток “Ruffier test”

В додатку є ряд недоліків, зокрема:

- в програмі немає контролю за правильністю виконання присідань;
- інтерфейс програми не є україномовним.

Основна проблема полягає в тому, що частота надто швидких або занадто повільних підйомів і присідань змінить кінцеве значення тесту. Для вирішення цієї проблеми розроблено додаток, який спростить проходження цього тесту і покаже користувачеві за допомогою анімації, з якою швидкістю слід виконувати присідання. Це допоможе підвищити точність введених користувачем даних і кінцевого результату.

Розділ.2 Опис використаних технологій

2.1 Microsoft Visual Studio 2019

Microsoft Visual Studio 2019 — це IDE, розроблена корпорацією Microsoft для різних видів розробки, а саме: настільних додатків, веб-додатків, веб-сервісів, веб-сайтів, мобільних додатків. Середовище містить інструменти для компіляції та полегшення процесу розробки [4].

Середовище Visual Studio дозволяє розробляти програми різними мовами програмування: Visual C#, Visual Basic, Visual F#, Visual C++, Python тощо (рис. 2). Також є можливість розробки додатків не тільки для Windows, але й для інших популярних платформ: Android, iOS.

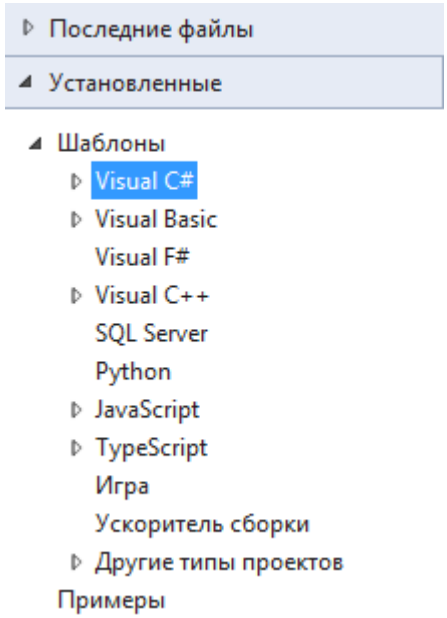

Рис. 2. Мови програмування, які підтримує VS

У Visual Studio є вбудований редактор коду, який підтримує технологію автозавершення Microsoft (IntelliSense) і рефакторинг коду (процес модифікації коду для покращення продуктивності продукту, щоб його було легко читати без зміни його поведінки) [5].

2.2 Мова програмування Python

Python - це дуже гнучка мова, її використовують для написання ігор, сервісів, веб-додатків, крон-скриптів для відновлення даних, тестування, а також з її допомогою навчають нейромережі [1].

Python дозволяє розбивати програми на модулі, що потім можуть бути використані в інших програмах. Python поставляється з великою бібліотекою стандартних модулів, які можна використовувати як основу для нових програм або як наприклад при вивченні мови [2]. Стандартні модулі надають засоби для роботи з файлами, системними викликами, мережевими з'єднаннями і навіть інтерфейсами до різних графічних бібліотек.

2.3 Kyvi

2.3.1 Створення базової програми на Python і Kivy

Установивши бібліотеку Kivy, ви можете створювати реляційні графічні інтерфейси, використовуючи лише Python і Kivy. Мови Python і Kivy дозволяють відокремити логіку програми від її представлення [3].

Приклад написання найпростішої програми Kivy за допомогою Python для відображення повідомлення «Hello world».

Щоб створити Kivy інтерфейс, нам потрібно імпортувати модуль програми app з Kivy.

У тілі програми клас FirstKivy успадковує клас App. Щоб створити вашу програму, функція build() має повернути графічний елемент. Цей приклад повертає текст «Hello Kivy».

Запуск програми на виконання:

```
FirstKivy().run()
```

Звичайно цей виклик розміщується у конструкції

```
if_name_== '_main_':
```

Зразки програм Python і Kivy (1.8.0). Програми Kiwi посилаються на батьківські теги <Labe> у Python програмі і присвоює його властивості text значення ’Hello’+’ World!’.

```
#FirstKivy.py
from kivy.app import App
from kivy.uix.label import Label
```

```
class
FirstKivy(Ap
p): def
build(self):
        return Label()

if_name_== '
    main_':
    FirstKivy().
    run()

#FirstKivy.kv
#:kivy 1.8.0

<Label>
      text: 'Hello'+' World!'
```

Результат виконання програми (рис. 3):

Рис. 3. Перша програма

Основною перевагою використання мови Kivy є те, що властивості графічних елементів можуть бути встановлені іншою програмою. Це значно спрощує код Python. У цьому відношенні Kivy схожий на стилі, що використовуються в CSS і HTML.

2.3.2 Графічні елементи Kivy

Елементи GUI Kivy називаються віджетами. Усі утиліти Kivy знаходяться в модулі kivy.uix. Щоб використовувати віджет, ви повинні спочатку імпортувати його, а потім створити його об'єкт [6]. У Kivy доступні такі віджети:

Таблиця 1 – Віджети Kivy

Label	Текст
Button	Кнопка
Checkbox	Квадрат з галочкою
Image	Зображення
Slider	Бігунок
Progress bar	Успішність
Text input	Поле введення
Toggle button	Перемикач
Switch	Двопозиційний перемикач
Drop-down list	Список що випадає
FileChooser	Вибір файлів
Popup	Допоміжний текст
Spinner	Випадаючий список значень

2.3.3 Графічний елемент “Надпис”

У першій програмі використовувався лише один графічний елемент — мітка. Щоб встановити розмір тексту, потрібно встановити розмір шрифту

за допомогою властивості font_size. Властивість визначається в конструкторі Label:

```
Label(text="Hello Label",
      font_size='100')
```

Вигляд надпису із заданим розміром шрифту (рис. 4):

Рис. 4. Вигляд шрифту

Ви також можете застосувати стилі до підписів, наприклад:

жирний, курсив, підкреслення та зміна кольору. Стилі задаються в атрибуті text за допомогою тегів, як і в мовах розмітки HTML. Наприклад, [u]…[/u] підкреслено, [b]…[/b] виділено жирним шрифтом, [i]…[/i] виділено курсивом, а [color]…[/color] є кольором визначення. Для цього також потрібно встановити для властивості розмітки значення True.

Надпис з такими заданими властивостями матиме вигляд (рис. 5):

Рис. 5. Властивості шрифту

2.3.4 Графічний елемент “Кнопка”

Зазвичай використовуваним графічним елементом є кнопка, яку можна натискати, активувати/дезактивувати, змінювати розмір і розташування, колір і легенду.

Щоб створити кнопку у формі, необхідно імпортувати клас Button.

```
from kivy.app import App
from kivy.uix.button import
Button class
FirstKivy(App):
```

```
    def build(self):
      return Button(text="Click!")

if_name_== '
    main_':
    FirstKivy().
    run()
```

Результат запуску програми показує, що вузол займає весь простір, оскільки його розмір не вказано. Щоб встановити розмір і положення кнопки, потрібно використовувати властивості size_hint і pos. Параметр pos визначає вихідну точку, розташовану в нижньому лівому куті вікна (рис. 6). Інші параметри вказуються в конструкторі кнопки через кому.

Якщо задати ці параметри в кнопці попереднього прикладу то її вигляд зміниться :

Рис. 6. Кнопка

Щоб встановити колір кнопки, необхідно використовувати властивість background_color. Перші три налаштування властивостей відповідають кольору RGB, а четверте — прозорості. Діапазон значень параметра від 0 до 1(рис. 7).

Кнопка із такими властивостями матиме вигляд :

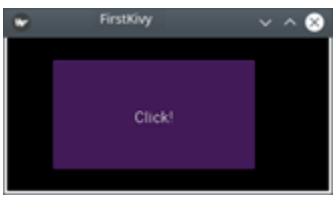

Рис. 7. кнопка з властивостями

Ви можете встановити активний або неактивний стан кнопки за допомогою властивості disabled. Якщо disabled=True, кнопка стане неактивною та темнішою, а клацання не відображатимуться.

2.3.5 Графічний елемент “Поле введення”

Поле введення, яке використовується для передачі даних від користувача до програми [7]. Щоб створити поле введення, вам потрібно імпортувати клас TextInput.

Зразок програми з графічною складовою «Поля введення»:

```
from kivy.app import App
from kivy.uix.textinput import
TextInput class FirstKivy(App):

    def build(self):
        return      TextInput(hint_text='Write      here',
                    size_hint=(.4,.1), pos=(250,250))

if_name_== '
    main_':
    FirstKivy().
    run()
```

Результат виконання програми (рис. 8):

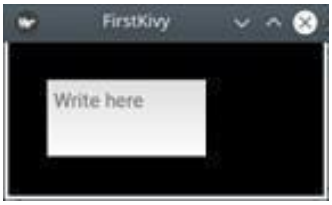

Рис. 8. Поле введення

2.4 Впорядкування елементів за допомогою контейнера layout

Layout – це контейнер, який використовується для розміщення графічних елементів певним способом:

AnchorLayout – положення елементів зверху, знизу, справа.

BoxLayout – розташувати елементи послідовно по вертикалі або горизонталі.

FloatLayout – розміщення елементів необмежене.

RelativeLayout – відносне розміщення елементів.

GridLayout – розміщення елементів у сітці, яка визначається властивостями rows і cols.

PageLayout – створення простих багатосторінкових розміщень для швидкого перемикання між сторінками.

ScatterLayout – розміщення подібне до RelativeLayout, але елементи можуть бути розтягнуті, повернуті і масштабовані.

StackLayout – поміщаються у стек зліва направо і зверху вниз.

У цьому прикладі спочатку створюється контейнер BoxLayout, а потім до нього додаються екземпляри кнопки [7]. Метод Constructor повертає BoxLayout з його внутрішніми елементами (рис. 9).

Результат виконання програми:

Рис. 9. Контейнер BoxLayout

Приклад розміщення елементів у контейнері GridLayout**.** Спочатку створюється контейнер сітка розміром 2×2, а потім додаються кнопки. Контейнер GridLayout підтримує властивості відступу padding і space.

Результат виконання програми (рис. 10):

Рис. 10. Контейнер GridLayout

Приклад розміщення елементів у контейнері AnchorLayout на фреймі контейнера. Контейнер AnchorLayout має два основні параметри, anchor_x і anchor_y, які відповідають положенню елемента вздовж горизонтальних і вертикальних країв контейнера. anchor_x може бути лівим, середнім, правим, а anchor_x може бути верхнім, середнім, нижнім (рис. 11)

Результат виконання програми :

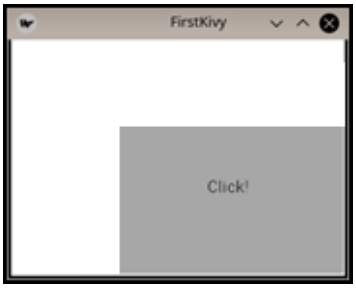

Рис. 11. Розміщення елементів у контейнері AnchorLayout

Щоб програма працювала, тобто виконувала певні дії, елементи GUI повинні реагувати на дії користувача, це робиться за допомогою подій. У Kivy майже для всіх графічних елементів є список подій, які викликаються певними діями користувача. Призначення певних подій елементу та прив’язування їх до певного методу оброблятиме події, що виконуються за допомогою функції bind(). Параметри елемента, незалежно від того, визначені вони в конструкторі елемента чи ні, можна змінювати програмно, наприклад, self.btn.text="Click!". Тут властивість text змінюється в об’єкті btn, а інші налаштування можна змінити таким же чином.

У цьому прикладі є лише один графічний елемент, кнопка заданого розміру та положення. За допомогою виклику self.btn.bind(on_press=self.btn_callback) подія on_press прив’язується до об’єкта кнопки, а метод btn_callback призначається цій властивості. Тепер при кожному натисканні кнопки буде викликаний метод btn_callback. Метод btn_callback приймає два аргументи: self і instance. У цьому випадку передається посилання на об'єкт, над яким була виконана та визначена дія. У тілі методу btn_callback властивість Instance.background_color об’єкта змінюється випадковим чином шляхом виклику random.random() для перших трьох аргументів (колір). Коли ви запустите цей приклад, ви побачите, що колір кнопки змінюється випадковим чином із кожним натисканням (рис. 13).

Результат виконання програми:

Рис. 13. Кнопка з змінюваним кольором.

Кнопка також може обробляти події on_release (коли кнопку відпущено) і стан (будь-яка зміна стану кнопки). Метод, який буде викликаний у події стану, отримає третій аргумент, стан кнопки. Кнопка може бути в двох станах - натиснута (вниз) і (ненатиснута (звичайне).

У цьому прикладі при кожній зміні стану викликається метод btn_callback, який відображатиме стан кнопки (значення параметра) на консолі.

При роботі з формами з кількома полями введення необхідно вказати роботу цих полів. Для цього поля введення обробляють текст і події фокусу.

Подія фокусування виникає, коли користувач запускає поле введення. Подія передає параметри self, instance і value зв'язаному методу. Параметр значення може мати два значення - True, якщо поле у фокусі, і False в іншому випадку.

self.txt_in.bind(focus=self.focus_callback)

Текстова подія виникає, коли змінюється вміст поля введення. Подія передає параметри self, instance і value, де значення є вмістом поля введення.

У цьому прикладі створюється поле введення та дві мітки, розташовані над і під полем введення. Події тексту та фокусу визначені для поля введення.

Текстова подія пов'язана з методом txt_callback, який призначає вміст поля введення заголовку. Оскільки подія запускається кожного разу, коли вміст змінюється, верхній індекс копіюватиме все, що введено в поле введення [3].

Подія focus пов'язана з методом focus_callback, який встановлює нижню мітку на «Focused» або «Out of focus» залежно від фокусу поля

(рис. 14).

Результат виконання програми:

Рис. 14. Поле введення із подіями.

Надписи також мають свого роду подію яка дозволяє зробити надписи інтерактивними. Для цього використовуються посилання в надписі, задані як стиль тексту. Текст виділений посиланням буде реагувати на натисканні на

ньому. Посилання в надписі задаються тегом [ref=variable]…[/ref].

Тут в надписі визначене посилання, яке закріплене за словом World, а до самого надпису за подією on_ref_press прив'язаний метод ref_click. Подія передаватиме в метод ref_click параметри self, instance і value яке буде дорівнюватиме змінній яка задана в тегу ref=world.

У цьому прикладі є дві мітки. Перший рядок містить слова "Hello World!" Від Kivy!", але кожне слово оточене посиланням, і в межах кожного посилання змінній призначається певна кількість слів. Подія on_ref_press встановлюється для першої реєстрації та до неї приєднується метод ref_click. Формула ref_click призначає текст «Натисніть слово {0}», яке використовує функцію format() замість {0}, значення, яке вставляє, щоб показати кількість натиснутих слів (рис. 15).

Результат виконання програми:

Рис. 15. Текст із посиланням.

2.3.6 Підтримка дат і часу в Kivy

Об'єкт Kivy Clock використовується для роботи з датами і часом. Для імпортування Kivy Clock потрібно включити наступний рядок [7].

Результат виконання програми (рис. 16):

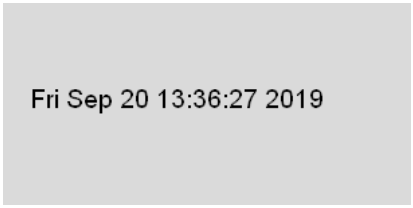

Рис. 16. Дата.

Підтримка часу в Kivy дозволяє викликати функції з певним інтервалом часу. Для цього ініціалізується об’єкт

Clock.schedule_interval(self.Clock_Callback, 1), де перший параметр визначає метод який буде викликатися, а другий – інтервал у секундах з яким буде викликатися метод.

Приклад програми, яка з інтервалом в одну секунду викликає метод, який при кожному виклику змінює надпис.

У класі FirstKivy додана змінна “i” яка буде лічильником. Імпортований та визначений Clock викликає метод Clock_Callback кожну секунду. Метод Clock_Callback збільшує змінну “i” на одиницю і змінює текст надпису на “Number of calls=”, в який підставляє значення змінної “i”.

Результат виконання програми (рис. 17):

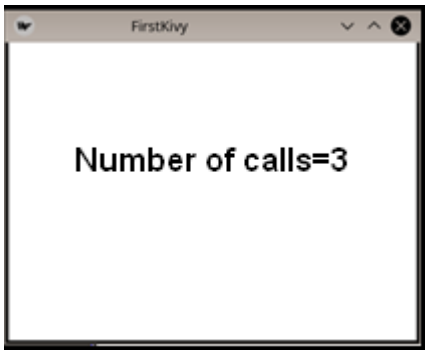

Рис. 17. Лічильник.

Висновки.

Бібліотека Kivy дозволяє створювати програми з сучасним графічним інтерфейсом засобами Python і Kivy майже для всіх платформ.

Розділ 3. Етапи створення додатка

Даний продукт був створений за допомогою мови програмування Python та кросплатформного фреймворку для графічних програм Kyvi. Перед початком розробки у середовищі встановлене необхідне програмне забезпечення а саме, Visual Studio 2019 для написання скриптів на мові Python.

3.1. Структура проекта

Після створення проект міститиме наступні файли (рис. 18). Файл main створюється за замовчуванням, інші файли були створені в процесі розробки.

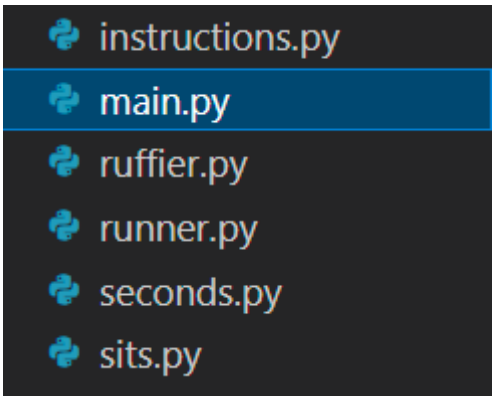

Рис.18. Структура проекта

- Instructions.py – містить необхідний інструкційний текст поміщений в змінні;
- Main.py – основний файл програми, містить підключення всіх необхідних бібліотек та модулів, описаний основний алгоритм роботи додатку;
- Ruffier.py – містить алгоритм який вираховує індекс та інтерпретує його, повертаючи рівень готовності серця;
- Runner.py – зберігає в собі алгоритм в якому описаний рух для анімованих фігур;

  Seconds.py – призначений для зберігання функцій підрахунку часу на різних етапах роботи програми;
- Sits.py – зберігає в собі функції для підрахунку кількості присідань;

3.2. Детальний опис функціоналу

Даний додаток складається з п’яти вікон, основне – це перше в якому міститься інструкція по використанню додатку. Основна ціль – полегшення

проходження тесту для користувача та підвищення точності даних. Додаток розроблений для операційної системи Android.

Користувач має виконати всі дії які його просить зробити програма і в результаті він має отримати текст, в якому буде описаний стан його серцево-судинної системи.

3.2.1. Вікно «Інструкції»

Перше що бачить користувач коли запускає додаток, це вікно з інструкцією та полями для введення потрібної інформації (рис. 19).

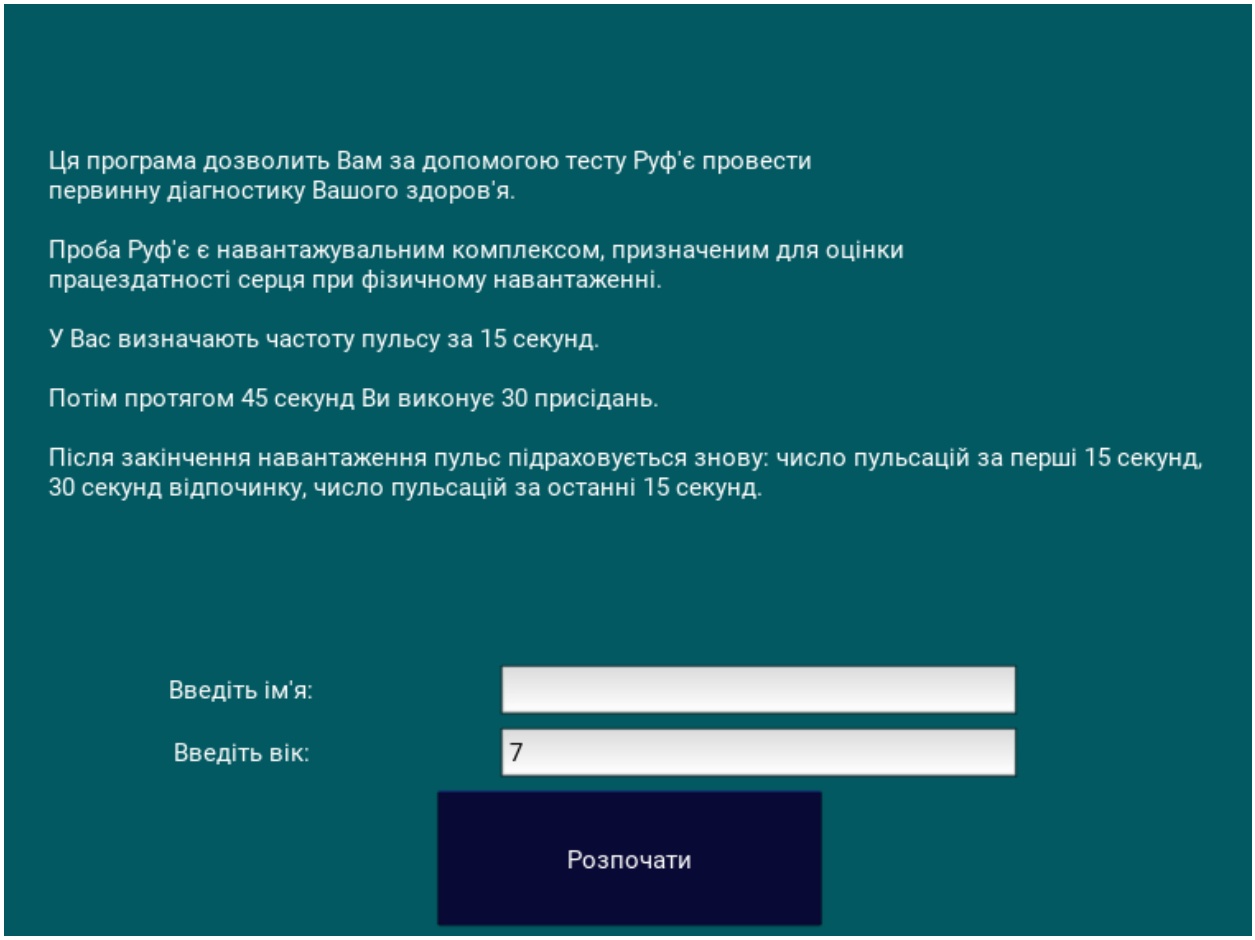

Рис.19. Вікно інструкції

Тут користувачеві потрібно ввести своє ім’я та вік. Вік потрібен тому що він є однією із змінних які використовуються для обчислення індекса Руф’є.

Для розробки цього вікна в файлі main.py було створено клас InstrScr, який є нащадком батьківського класу Screen із бібліотеки Kivy. Детальніше можна

глянути в додатку 2. В класі описаний конструктор в якому в якому прописані властивості цього вікна.

Після створення порожнього вінка потрібно було створити слої і на ці слої розмістити всі необхідні об’єкти які мають бути у цьому вікні. Створено два слоя, на перший поміщено текст інструкції (рис. 20).

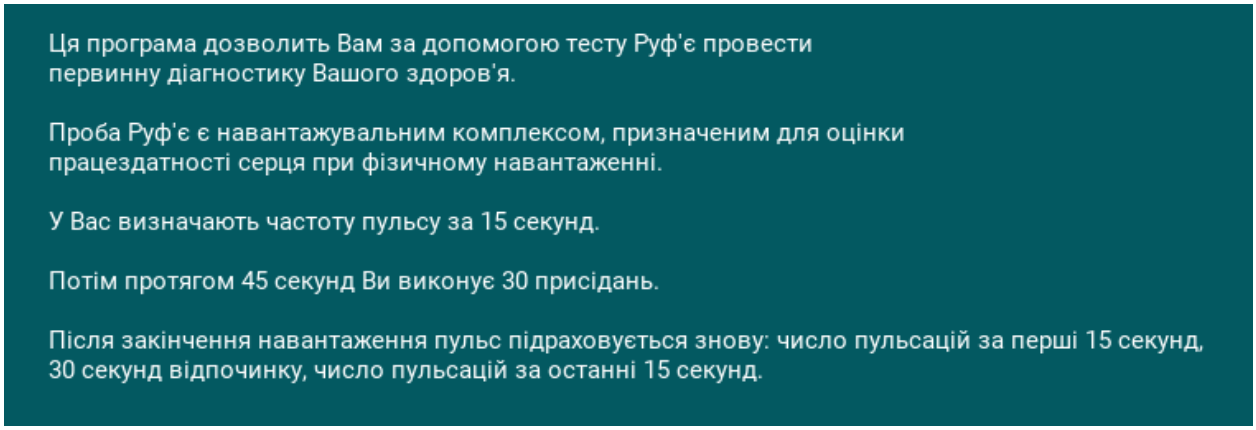

Рис.20. Інструкція

Весь текст який інструкцій, який використовується для того щоб вказати користувачу що йому потрібно робити описаний та зберігається в змінних, які містяться в файлі instructions.py. Детальніше можна глянути в додатку 1. Для того щоб можна було його використовувати, до файлу main.py було підключено файл який містить текст, за допомогою функції import. Детальніше можна глянути в додатку 2. На другому розмістилися поля для введення віку та імені, їх текст та кнопка з написом “Розпочати” (рис. 21).

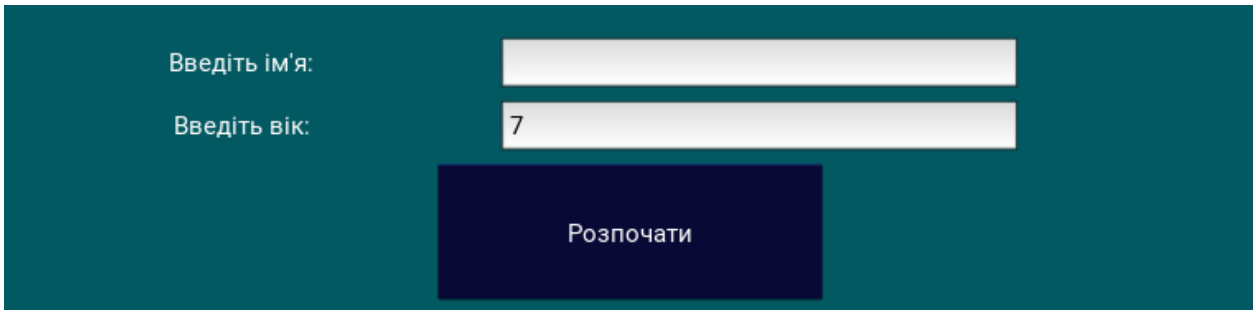

Рис.21. Поле введення

Також була розроблена функція яка перевіряє введені дані користувачем, а саме перевіряє який вік він ввів. Якщо введені дані можуть перетворитися в числовий формат цілого числа int, тоді програма дозволяє продовжити

роботу. Цей алгоритм був написаний за допомогою функції try. Якщо користувач вводить дані які не можуть перетворитися в ціле число, тоді програма видаляє написані дані, і заповнює форму стандартним віком. Приклад цього можна побачити на (рис. 22) та (рис. 23).

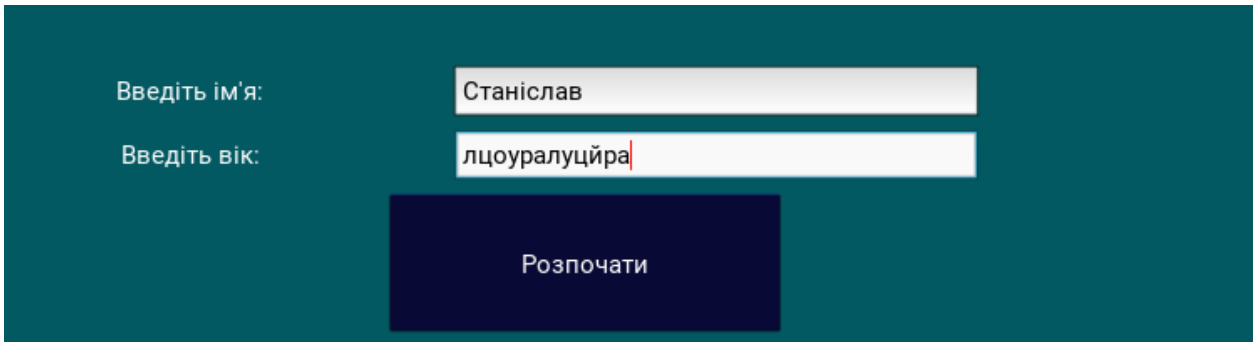

Рис.22. Невірно введені дані

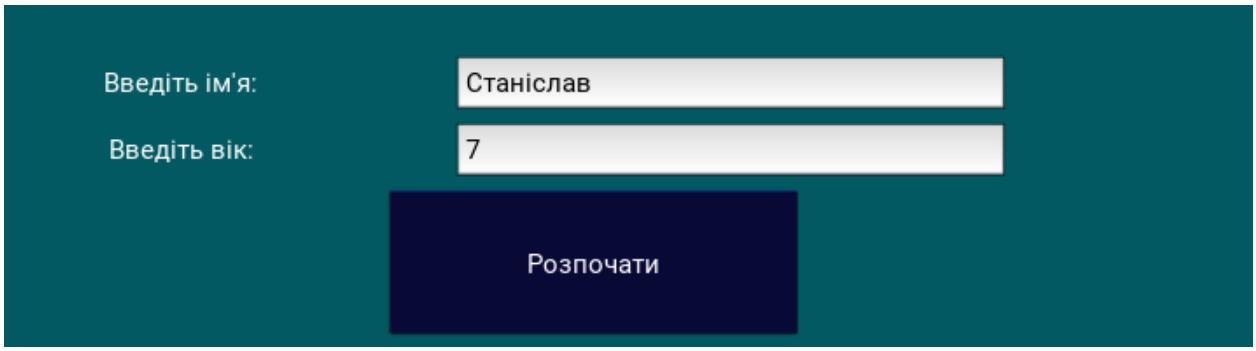

Рис.23. Автозаміна невірних даних

Якщо користувач ввів дані вірно, тоді дані зберігаються і відбувається перехід на наступне вікно.

Також потрібно було задати всім об’єктам певний колір. Для цього було створено змінні які містили в собі чотири параметри. Перші три параметри властивості відповідають кольорам RGB, а четвертий – прозорості. Діапазон значень параметрів від 0 до 1 (рис. 24).

```
Window.clearcolor = (0.01, 0.35, 0.38, 1)
btn_color = (0.10, 0.10, 0.60, 1)
```

Рис.24. Кольори

3.2.2. Вікно “Пульс”

Для того щоб виміряти індекс нам потрібно знати три числа. Перше число це кількість ударів яке зробило серце користувача за 15 секунд в стані спокою. Саме цю інформацію має внести користувач в другому вікні (рис. 25).

Рис.25. Вікно “Пульс”

На цьому вікні є три слої. На першому розміщений текст інструкції, який імпортований з файлу instructions.py. Детальніше можна глянути в додатку 1. На другому слої міститься текст із таймером, який рахуватиме кількість секунд. На третьому розміщено поле для введення даних та кнопка почати. Для розробки цього вікна в файлі main.py було створено клас PulseScr, який є нащадком батьківського класу Screen із бібліотеки Kivy. Детальніше можна глянути в додатку 2.

Коли користувач натискає на кнопку, запускається функція, яка відраховує 15 секунд. Поки цей час не пройде, користувач не може вводити дані та

перейти на наступний етап. Поле для вводи залишається неактивним, а кнопка змінює свою прозорість, цим самим допомагає користувачу зрозуміти що вона не активна (рис. 26).

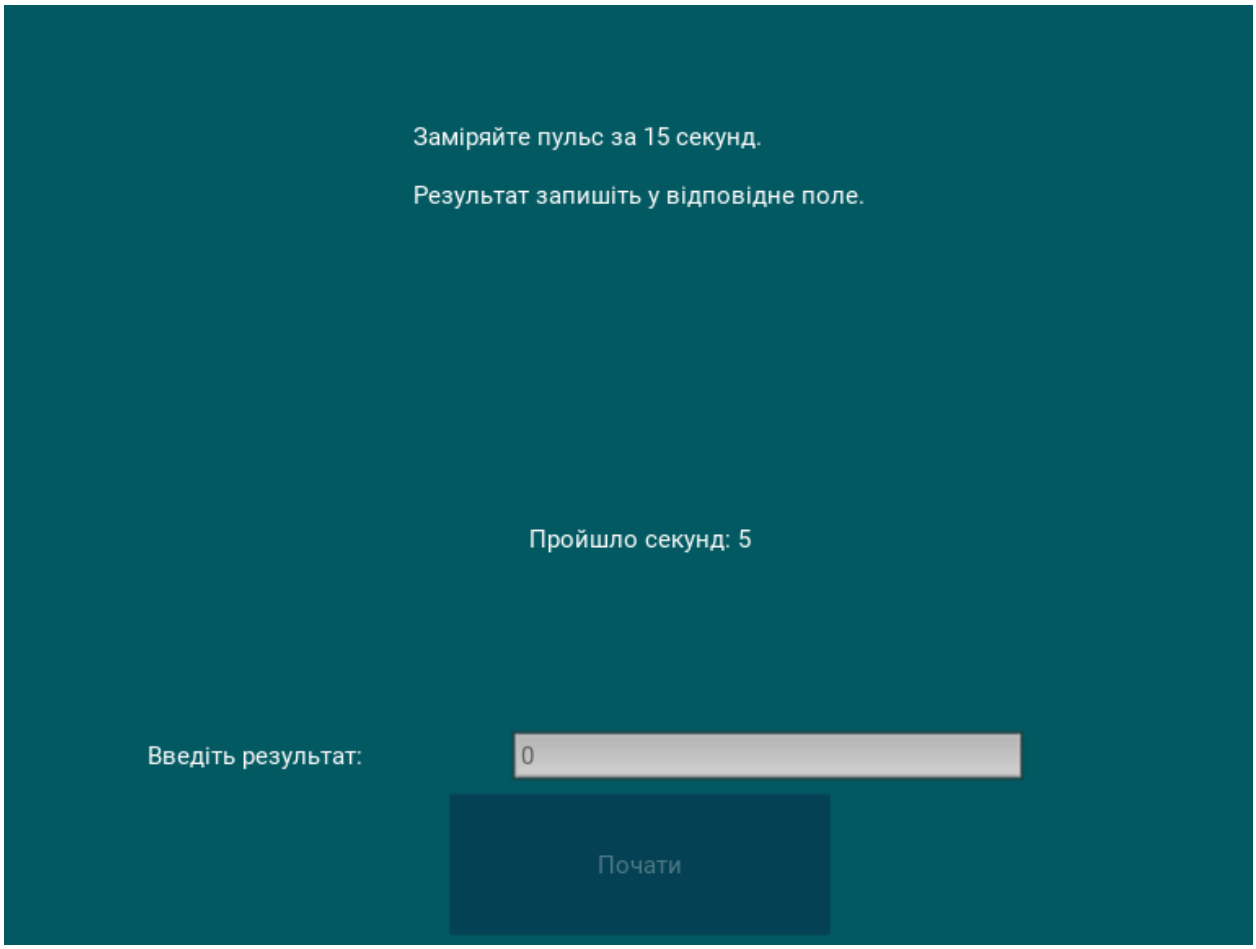

Рис.26. Робота програми після натиску на кнопку

Під час того як йде відлік часу, користувач має рахувати кількість ударів його серця. Після закінчення відліку поле для введення даних стає активним, а кнопка змінює свою прозорість та текст (рис. 27).

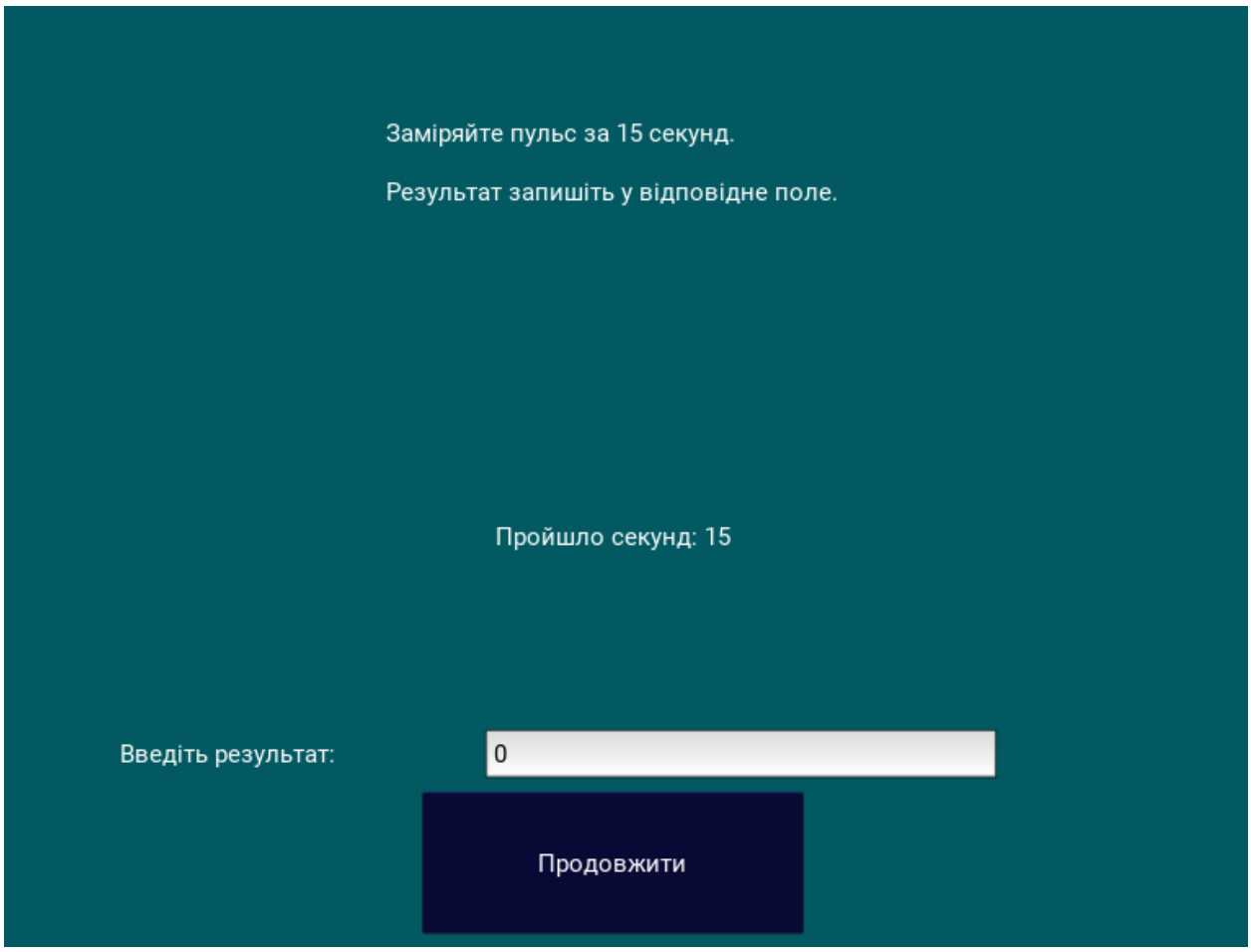

Рис.27. Введення першого результату

Для поля введення інформації була розроблена функція, яка перевіряє введені користувачем дані. Якщо користувач впише не числову інформацію, або це число буде менше чи рівне 0, тоді форма очиститься і заповниться стандартним числом 0, тим самим дає користувачу зрозуміти, що він ввів неправильні дані. Після введення коректних даних, користувач перейде до третього вікна.

3.2.3. Вікно “Присідання”

Третє вікно є одним із самих важливих, тому що тут користувачу потрібно буде виконувати фізичне навантаження. На цьому вікні є три слоя. На першому слої розташований прямокутник багряного кольору. На другому слої інструкція, таймер часу, лічильник присідань. На третьому кнопка почати.

Для розробки цього вікна в файлі main.py було створено клас CheckSits , який є нащадком батьківського класу Screen із бібліотеки Kivy (рис. 28). Детальніше можна глянути в додатку 2.

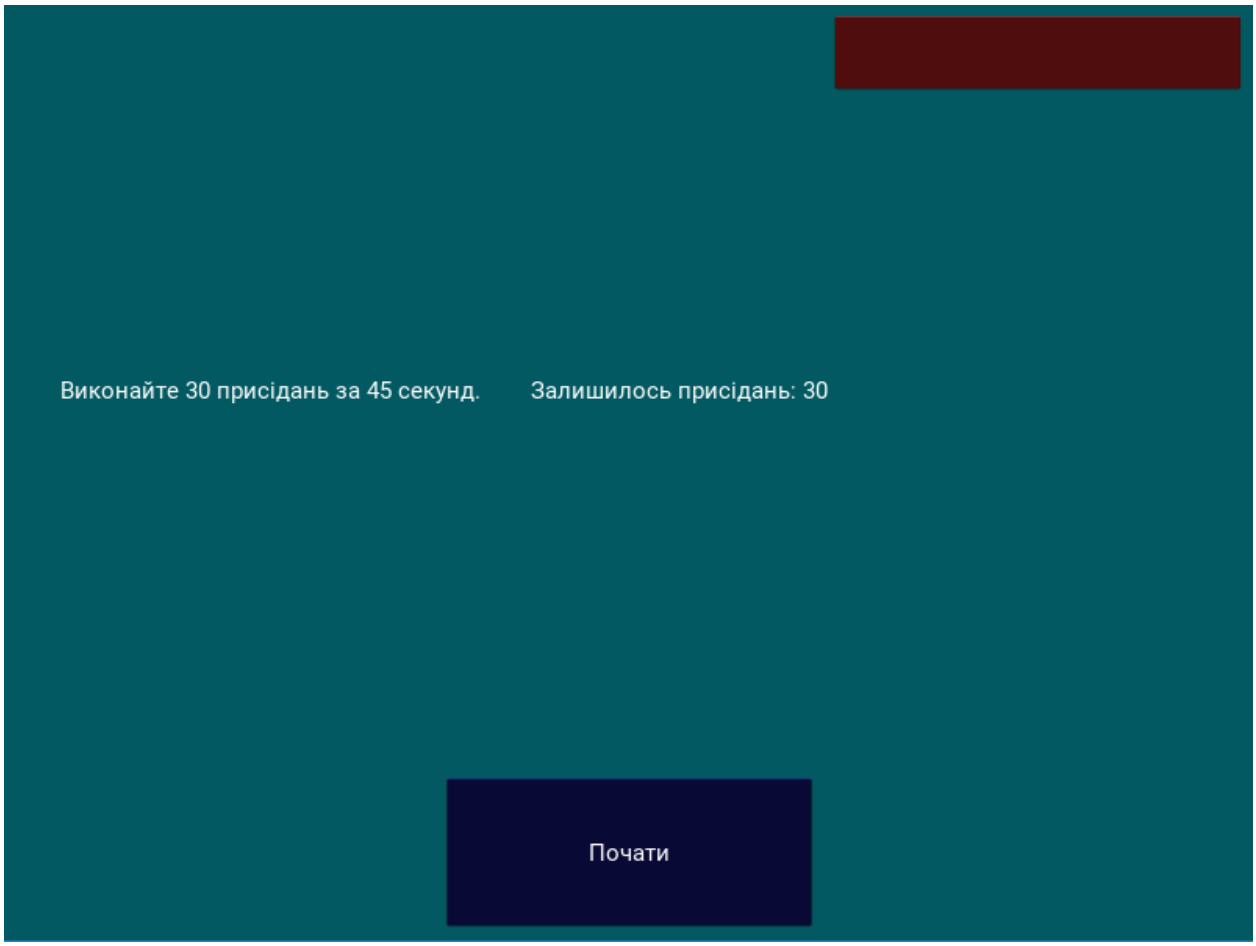

Рис.28. Вікно “Присідання”

Прямокутник був створений для того щоб вирішити основну проблему неточності даних, а саме не правильний темп присідань. Для того щоб дані були максимально точними користувачу потрібно виконувати одне присідання в півтори секунди. При цьому його рухи мають бути плавними. Для цього була розроблена анімація для прямокутника багряного кольору.

Коли користувач натискає на кнопку почати, йому потрібно присідати і його такт присідань має бути таким самим як в прямокутника. Також після натискання на кнопку запускається таймер який рахуватиме сорок секунд та лічильник, який рахує кількість присідань (рис. 29).

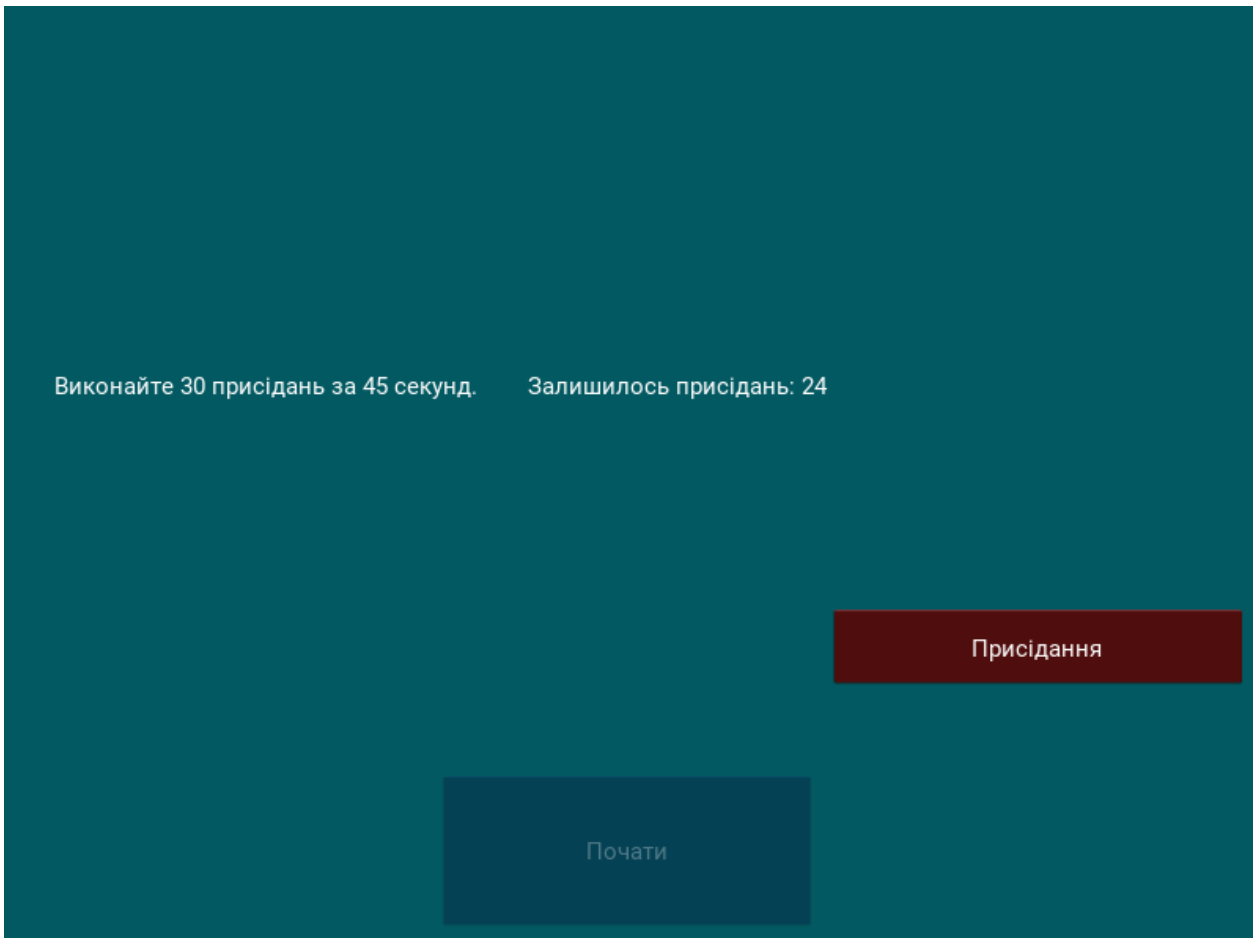

Рис.29. Анімація присідань

Алгоритм роботи таймерів описаний в файлі seconds.py, а алгоритм підрахунку кількості присідань описані в файлі sits.py. Детальніше можна глянути в додатку 5 та додатку 6. Присідання зараховується тоді, коли анімація прямокутника зробила два повтори. Тобто він виконав рух вниз і повернувся знову наверх. В такому випадку кількість змінюється.

Після виконання фізичних навантажень, користувачеві потрібно натиснути на кнопку продовжити і він перейде до четвертого екрану (рис. 30).

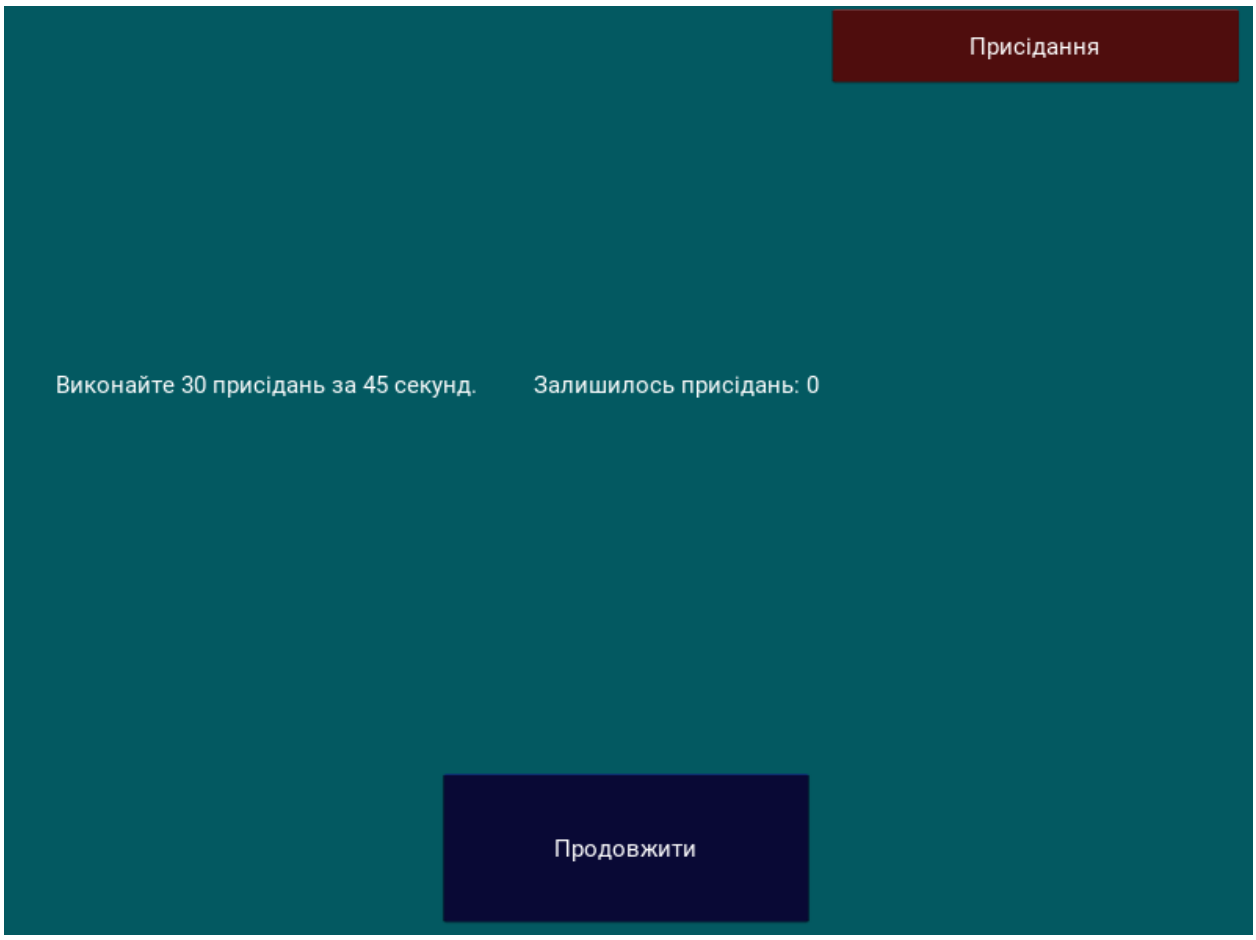

Рис.30. Завершення присідань

3.2.4. Вікно “Пульс 2”

На четвертому вікні є чотири слої. На яких розміщені текст інструкції, таймер часу та поля для введення даних, відповідно. Для розробки цього вікна в файлі main.py було створено класPulseScr2, який є нащадком батьківського класу Screen із бібліотеки Kivy (рис. 31).

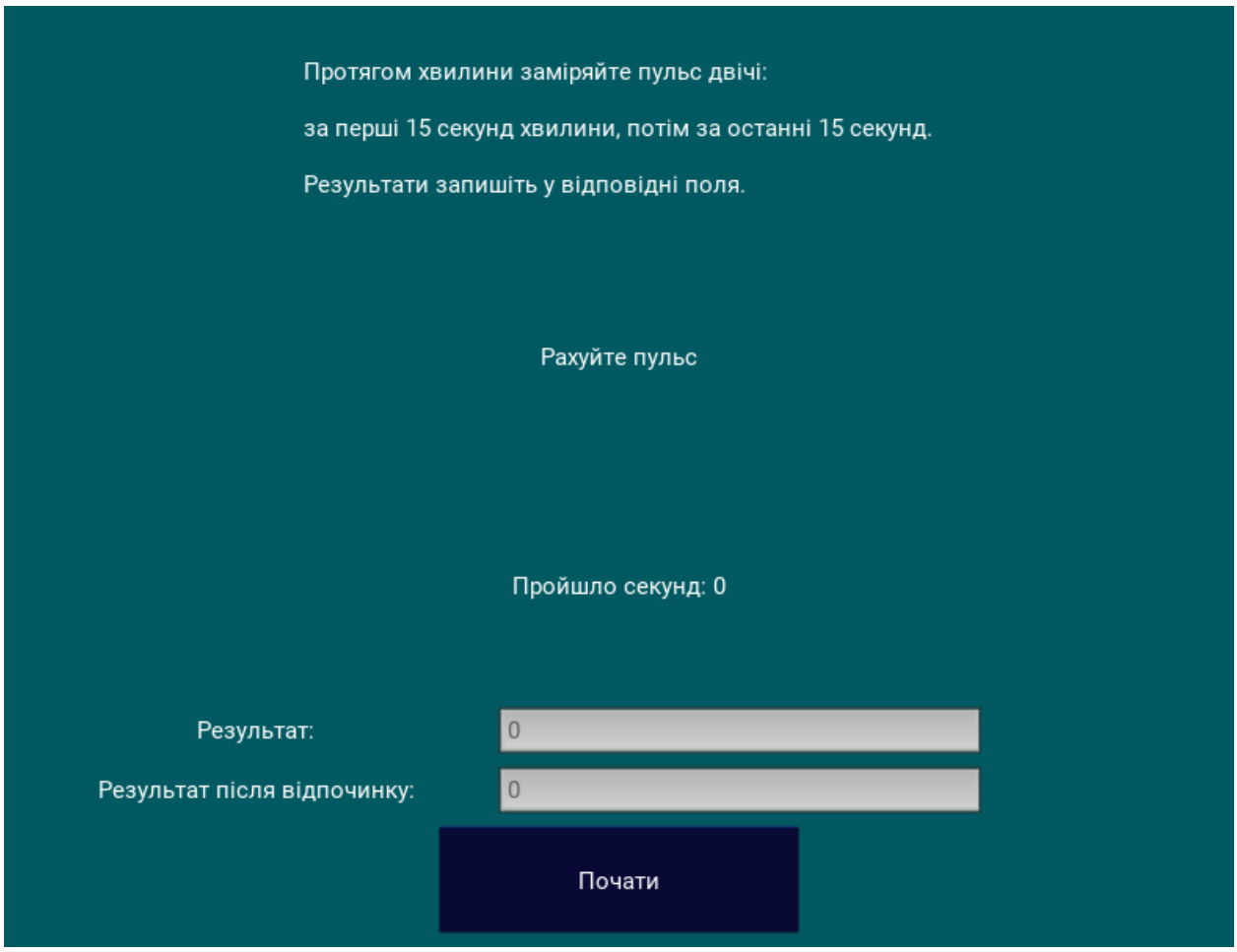

Рис.31. Вікно “Пульс 2”

Тут користувачеві потрібно буде порахувати кількість ударів серця після фізичного навантаження. Натиснувши на кнопку “Почати” запуститься таймер який відраховує 15 секунд. Цей час потрібен для того щоб порахувати кількість ударів серця (рис. 32). Під час того як триває відлік, кнопка та поля для введення стають неактивні.

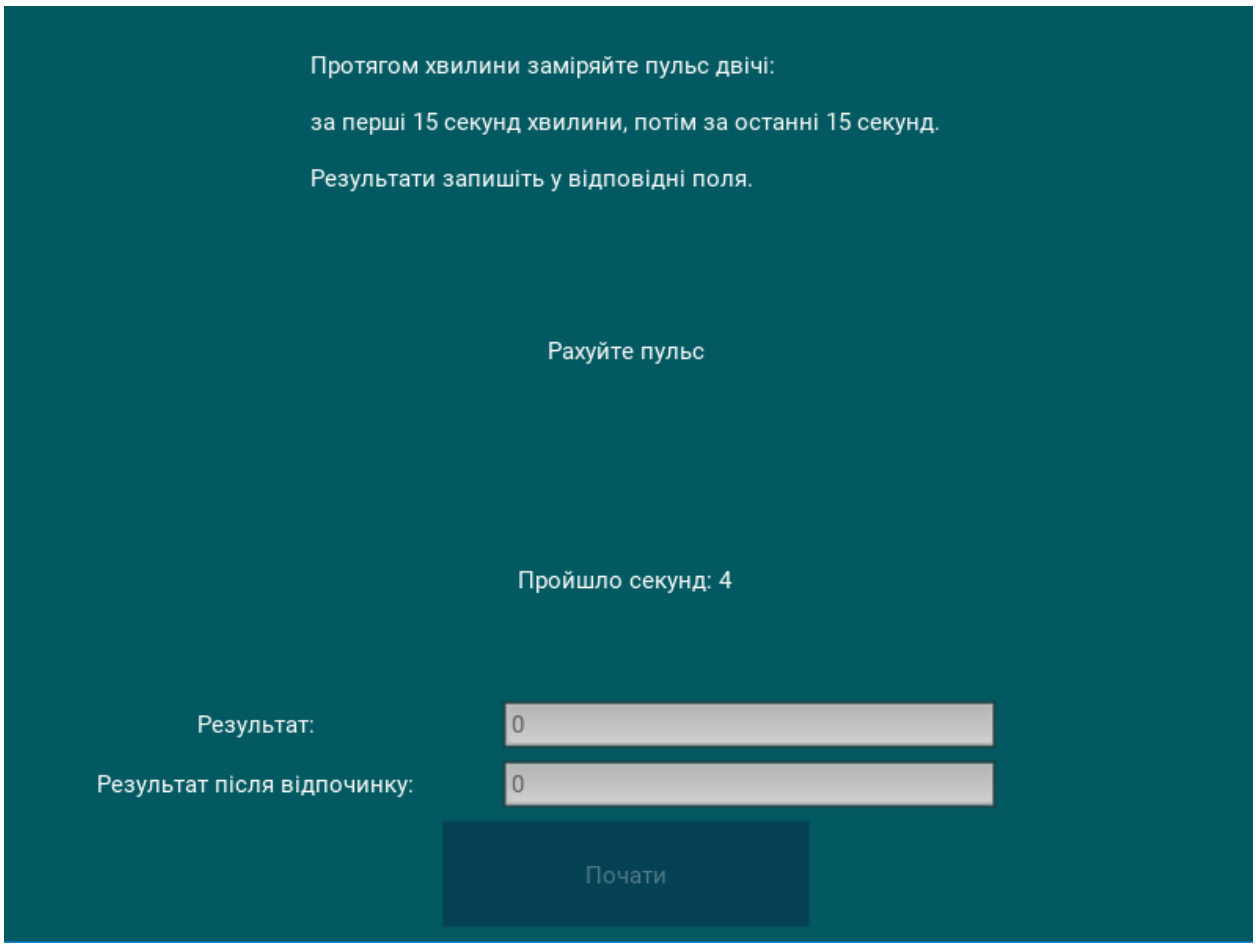

Рис.32. Робота таймера

Після закінчення відліку, поле для введення “Результат” стає активним і користувач може ввести туди свої дані. Також для зручності користувача було додану підказку в центрі екрану. Цей текст допомагає зрозуміти що потрібно робити в даний момент. Коли тривав відлік часу, було написано “Рахуйте пульс”.

Після закінчення перших п’ятнадцяти секунд, запускається наступний таймер, а саме на тридцять секунд. Цей таймер потрібен для відпочинку. Під час роботи цього таймера, користувач бачить текст “Відпочивайте” (рис. 33).

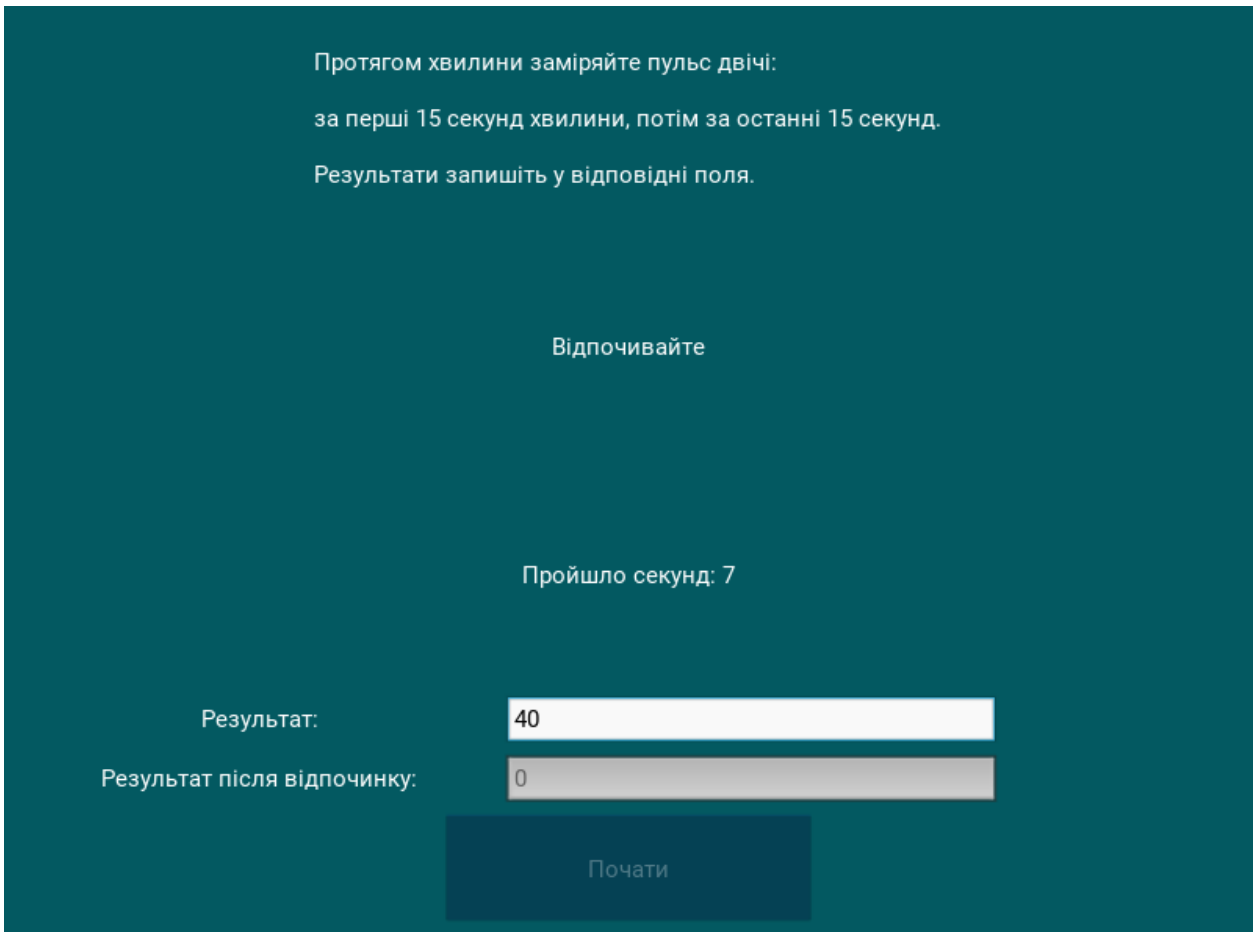

Рис.33. Введення другого результату

Після того як пройшло тридцять секунд відпочинку користувач знову має почати рахувати пульс. Текст змінюється на “Рахуйте пульс”, таймер починає відлік п’ятнадцяти секунд. Після закінчення роботи таймера поле для введення тексту “Результат після відпочинку” стає активним і користувач може внести свої дані. Для введених користувачем даних, також була написана функція, яка перевіряє що він ввів і якщо ці дані не є коректними, то програма їх видаляє та заповнює стандартним значенням нуль. Це допомагає користувачеві зрозуміти що він ввів неправильні дані. Кнопка “Почати” стає активною та змінює напис на “Завершити”. Після натискання на кнопку користувач перейде на останнє вікно, яке покаже йому його результат (рис. 34).

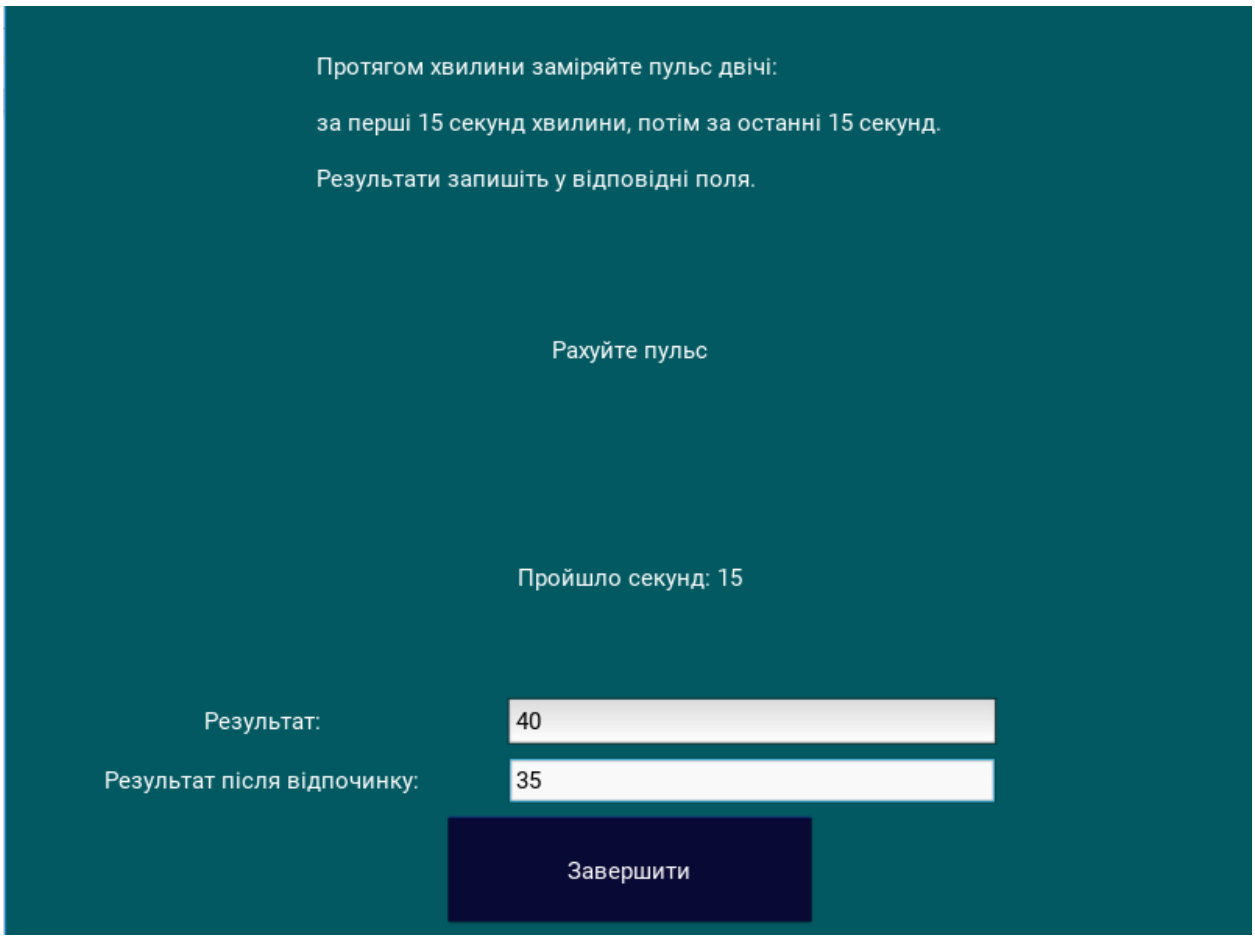

Рис.34. Введення останнього результату

3.2.5. Вікно “Результати”

На цьому вікні є всього один слой, на якому виводиться текст з результатами обчислень. Програма вираховує індекс Руф’є і в залежності від нього виводить користувачеві заготовлений текст (рис. 35).

Рис.35. Фінальне вікно

В залежності від індексу, текст може бути таким:

- "Працездатність серця: низька. Терміново зверніться до лікаря!"
- "Працездатність серця: задовільна. Зверніться до лікаря!"
- "Працездатність серця: середня. Можливо, варто додатково обстежитись у лікаря."
- "Працездатність серця: вище середнього"
- "Працездатність серця: висока"

Розрахунок результатів проби прописано в модулі ruffier.py. Детальніше можна глянути в додатку 3.

Сума вимірювань пульсу у трьох спробах (до навантаження, одразу після та після короткого відпочинку) в ідеалі має бути не більше 200 ударів на хвилину. Ми пропонуємо дітям вимірювати свій пульс протягом 15 секунд, і наводимо результат до ударів за хвилину множенням на 4:

$$S = 4* (P1 + P2 + P3)$$

Що далі цей результат від ідеальних 200 ударів, то гірше. Традиційно таблиці даються для величини, поділеної на 10.

Індекс Руф'є

$$IR = (S - 200) / 10$$

оцінюється за таблицею відповідно до віку:

	7-8	9-10	11-12	13-14	15+(тільки для підлітків)
чудово	6.4 і менше	4.9 і менше	3.4 і менше	1.9 і менше	0.4 і менше
добре	6.5 - 11.9	5 - 10.4	3.5 - 8.9	2 - 7.4	0.5 - 5.9
задовільно	12 - 16.9	10.5 - 15.4	9 - 13.9	7.5 - 12.4	6 - 10.9
слабкий	17 - 20.9	15.5 - 19.4	14 - 17.9	12.5 - 16.4	11 - 14.9
незадовільний	21 і більше	19.5 і більше	18 і більше	16.5 і більше	15 і більше

Для всіх вікових груп результат "незадовільний" відрізняється від "слабкий" на 4 одиниці, той від "задовільно" на 5 одиниць, а "добрий" від "задовільно" - на 5.5.

S = p1 + p2 +p3, де p1, p2, p3 це дані серцебиття, які вводив користувач під час роботи програми відповідно.

Функція ruffier_result отримує на вхід індекс Руф’є і інтерпретує його, повертає рівень готовності: число від 0 до 4 (що вище рівень готовності, то краще).

Функція test повертає готові тексти, які залишається намалювати у потрібному місці. Використовує для текстів константи, задані на початку цього модуля.

3.3. Тестування роботи додатку

А зараз я протестую роботу додатку, буду виконувати всі дії та вносити дані. Для початку впишемо моє ім’я та вік (рис. 36).

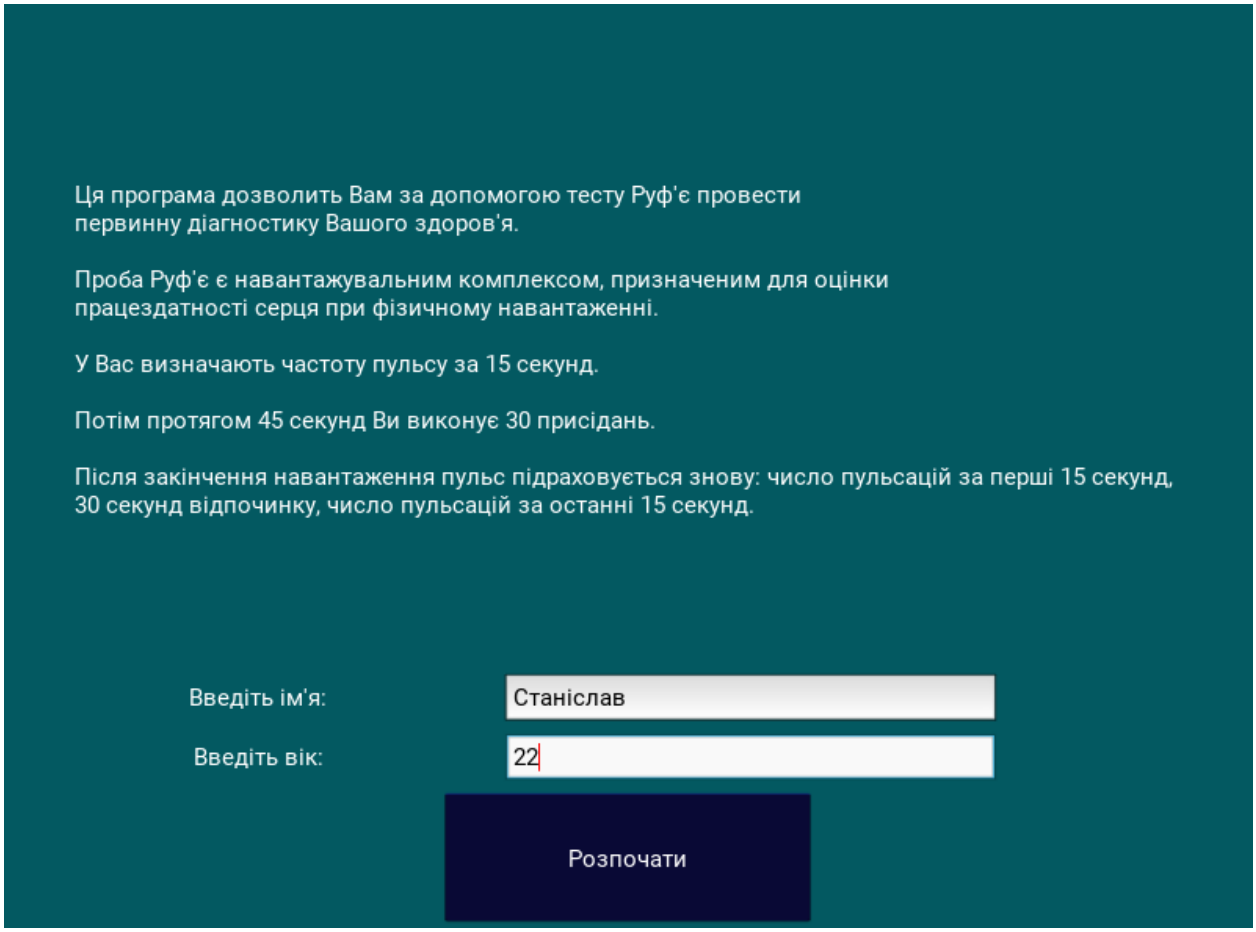

Рис.36. Введення початкових даних

Далі рахуємо кількість ударів серця за 15 секунд, в мене це 20 ударів (рис. 37).

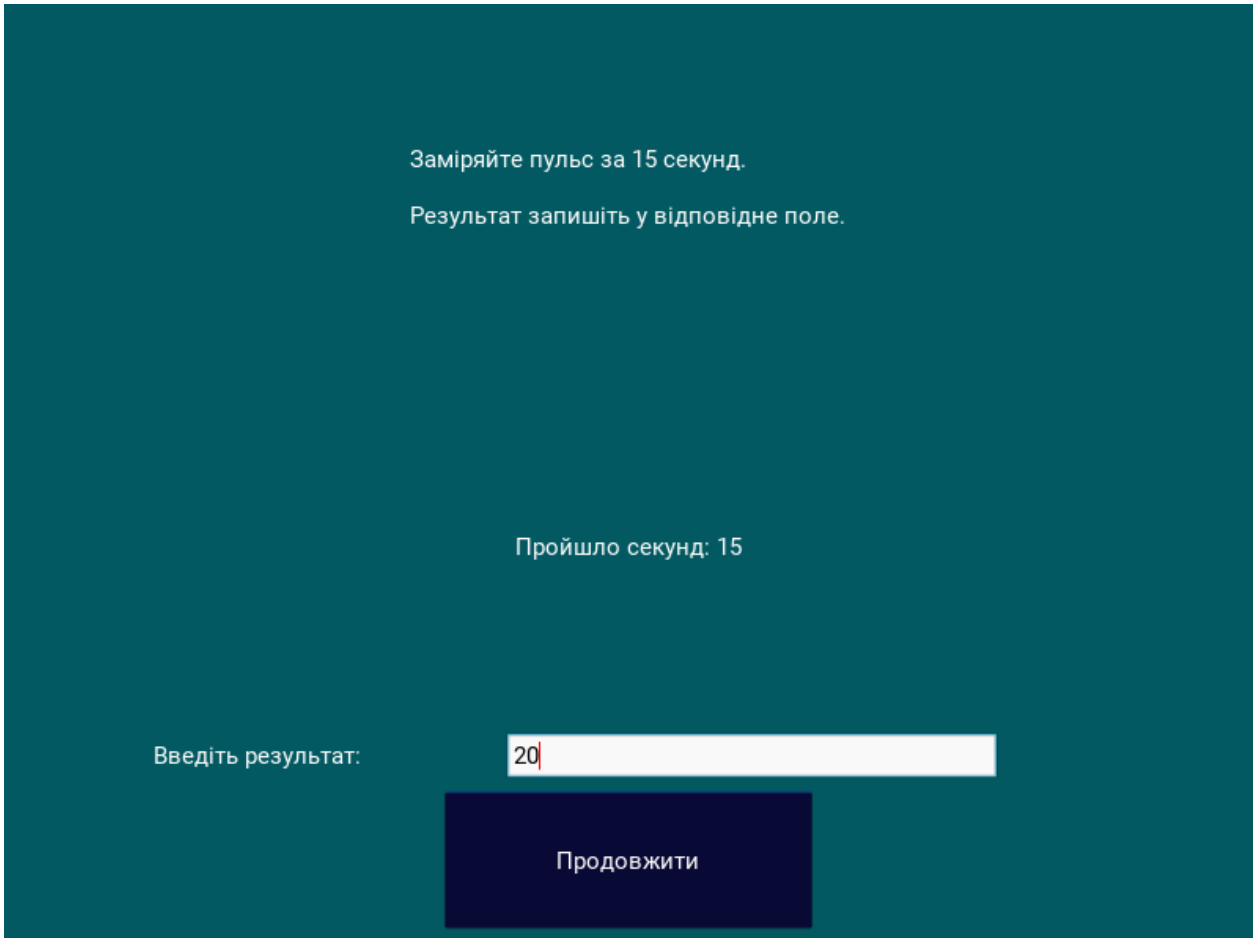

Рис.37. Введення першого результату

Після цього я виконую 30 присідань за 45 секунд. Після цього починаю рахувати свій пульс та вносити дані слідуючи інструкції програми. Мої результати після присідань це 31 удар, і після 30 секунд відпочинку це 20 дарів (рис. 38).

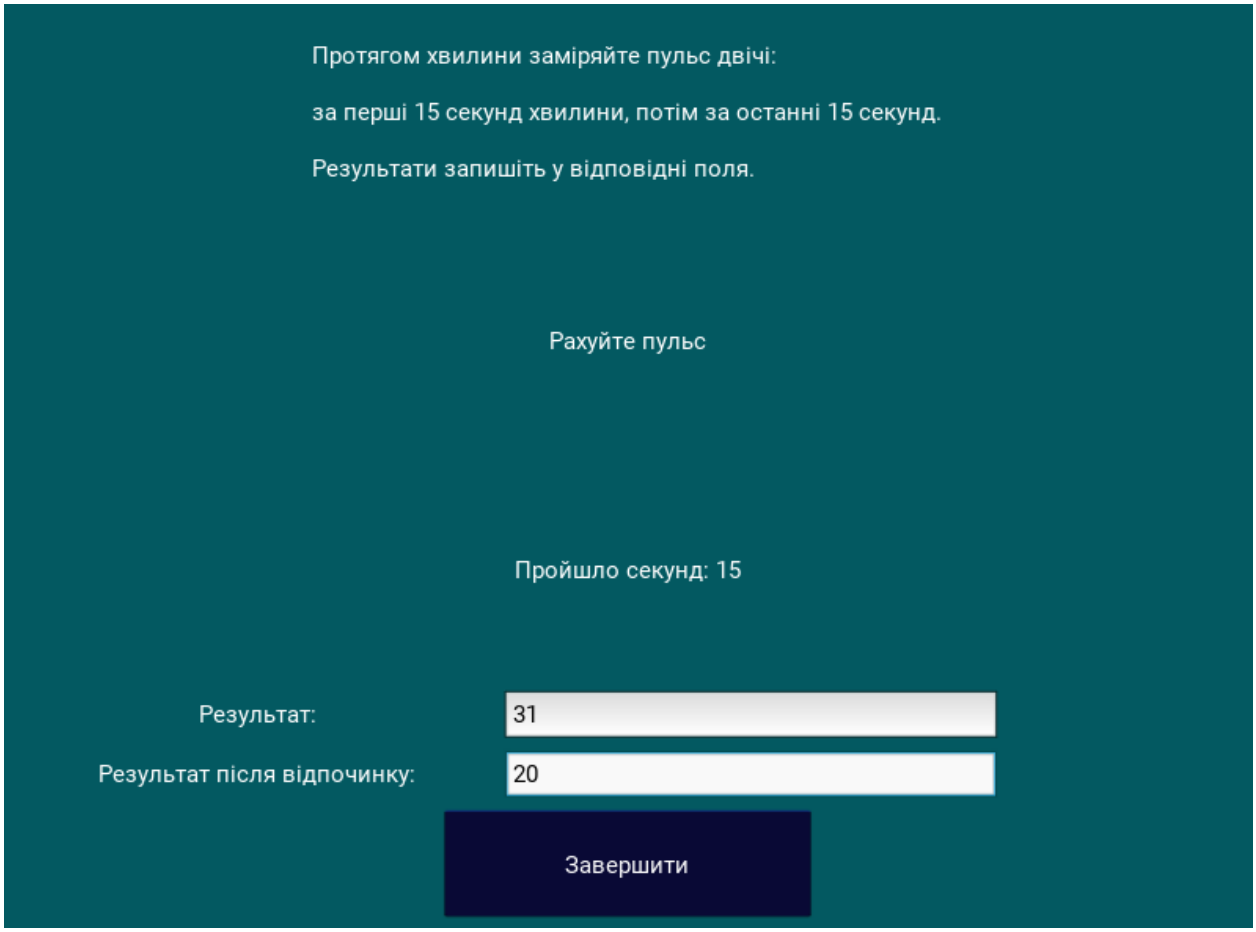

Рис.38. Введення останніх результатів

Ось мій результат (рис. 39).

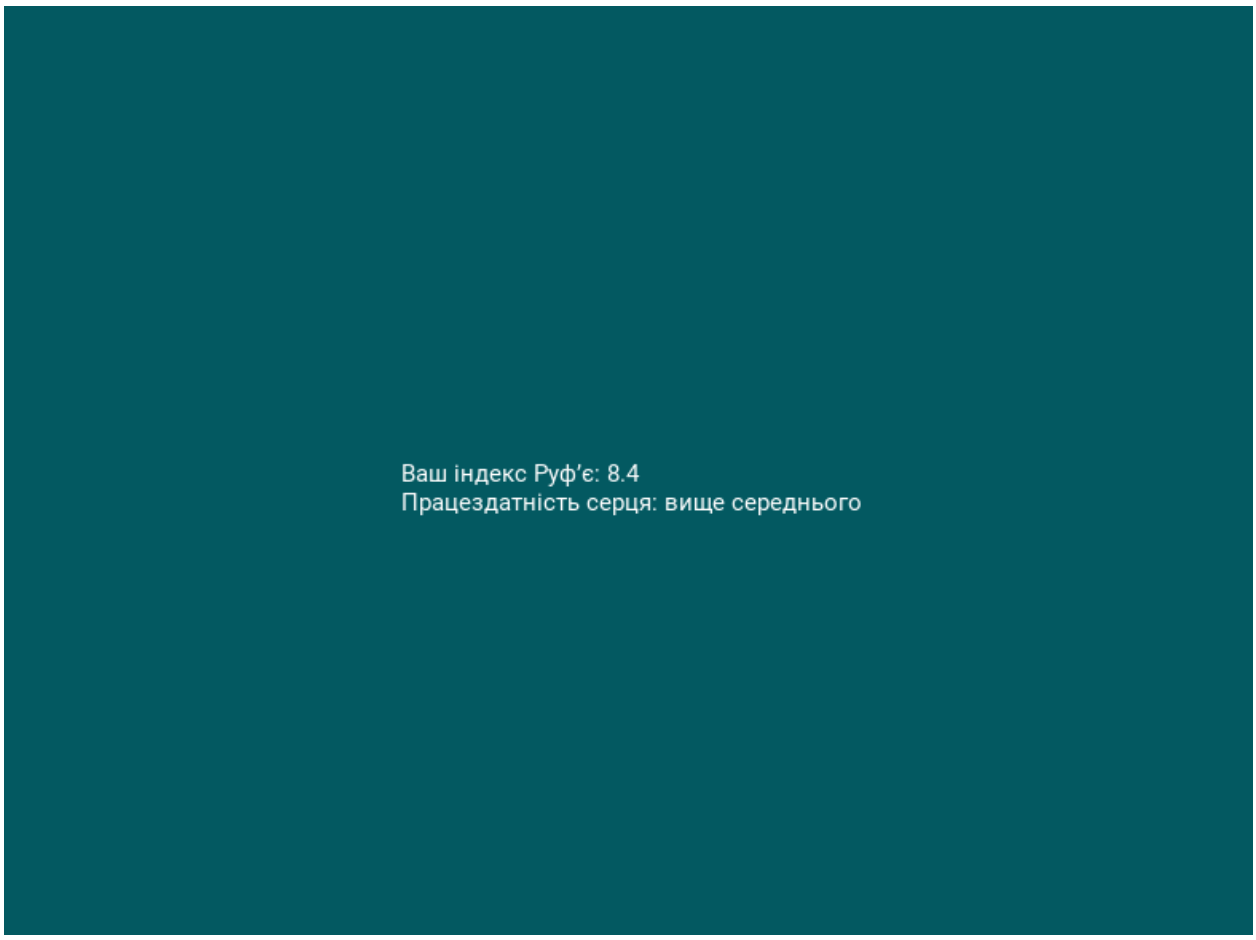

Рис.39. Мій результат

Висновок

В результаті було створено кросплатформенне програмне забезпечення під операційну систему Windows та планується адаптація під систему Android, яке дозволяє пройти тестування та дізнатися точний результат свого індексу Руф’є. Додаток може використовуватися як в медичних установах, школах на уроці фізкультури, так і для особистого використання, щоб для себе зрозуміти чи потрібно звертатися до лікаря, чи ні.

СПИСОК ДЖЕРЕЛ ТА ЛІТЕРАТУРИ

1. What is Python? [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/doc/essays/blurb/.
2. Керівництво по Python. [Електронний ресурс] – Режим доступу до ресурсу: https://python-scripts.com/.
3. Документація Kivy. [Електронний ресурс] – Режим доступу до ресурсу: https://kivy.org/doc/stable/.
4. Visual Studio. [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/.
5. Getting started with Python in Visual Studio. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/training/modules/python-install-vscode/.
6. Навчальний курс Kivy. [Електронний ресурс] – Режим доступу до ресурсу: https://kivycoder.com/.
7. Kivy Tutorial. [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/kivy-tutorial/.
8. Stackoverflow. [Електронний ресурс] – Режим доступу до ресурсу: https://stackoverflow.com/

ДОДАТКИ

Додаток 1. Файл instructions.py:

```
txt_instruction = '''
Ця програма дозволить Вам за допомогою тесту Руф'є провести
первинну діагностику Вашого здоров'я.\n
Проба Руф'є є навантажувальним комплексом, призначеним для оцінки
працездатності серця при фізичному навантаженні.\n
У Вас визначають частоту пульсу за 15 секунд.\n
Потім протягом 45 секунд Ви виконує 30 присідань.\n
Після закінчення навантаження пульс підраховується знову: число пульсацій за перші 15 секунд,
30 секунд відпочинку, число пульсацій за останні 15 секунд.'''

txt_test1 = '''Заміряйте пульс за 15 секунд.\n
Результат запишіть у відповідне поле.'''

txt_test2 = '''Виконайте 30 присідань за 45 секунд.\n
Натисніть кнопку "Почати", щоб запустити лічильник присідань.\n
Робіть присідання зі швидкістю лічильника.'''

txt_test3 = '''Протягом хвилини заміряйте пульс двічі:\n
за перші 15 секунд хвилини, потім за останні 15 секунд.\n
Результати запишіть у відповідні поля.'''

txt_sits = 'Виконайте 30 присідань за 45 секунд.'
```

Додаток 2. Файл main.py:

```
from kivy.app import App
from kivy.uix.screenmanager import ScreenManager, Screen
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.textinput import TextInput
from kivy.core.window import Window
from kivy.uix.scrollview import ScrollView
from instructions import txt_instruction, txt_test1, txt_test2, txt_test3, txt_sits
from ruffier import test

from seconds import Seconds
from sits import Sits
from runner import Runner

Window.clearcolor = (0.01, 0.35, 0.38, 1)
btn_color = (0.10, 0.10, 0.60, 1)

age = 7
name = ""
p1, p2, p3 = 0, 0, 0
```

```
def check_int(str_num):
    # повертає число або False, якщо рядок не конвертується
    try:
        return int(str_num)
    except:
        return False

class InstrScr(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        instr = Label(text=txt_instruction)
        lbl1 = Label(text="Введіть ім'я:", halign='center')
        self.in_name = TextInput(multiline=False)
        lbl2 = Label(text='Введіть вік:', halign='center')
        self.in_age = TextInput(text='7', multiline=False)
        self.btn = Button(text='Розпочати', size_hint=(0.3, 0.2), pos_hint={'center_x': 0.5})
        self.btn.background_color = btn_color
        self.btn.on_press = self.next
        line1 = BoxLayout(size_hint=(0.8, None), height='30sp')
        line2 = BoxLayout(size_hint=(0.8, None), height='30sp')
        line1.add_widget(lbl1)
        line1.add_widget(self.in_name)
        line2.add_widget(lbl2)
        line2.add_widget(self.in_age)
        outer = BoxLayout(orientation='vertical', padding=8, spacing=8)
        outer.add_widget(instr)
        outer.add_widget(line1)
        outer.add_widget(line2)
        outer.add_widget(self.btn)
        self.add_widget(outer)
    def next(self):
        name = self.in_name.text
        age = check_int(self.in_age.text)
        if age == False or age < 7:
            age = 7
            self.in_age.text = str(age)
        else:
            self.manager.current = 'pulse1'
class PulseScr(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.next_screen = False

        instr = Label(text=txt_test1)
        self.lbl_sec = Seconds(15)
        self.lbl_sec.bind(done=self.sec_finished)

        line = BoxLayout(size_hint=(0.8, None), height='30sp')
        lbl_result = Label(text='Введіть результат:', halign='right')
        self.in_result = TextInput(text='0', multiline=False)
```

```
        self.in_result.set_disabled(True)

        line.add_widget(lbl_result)
        line.add_widget(self.in_result)
        self.btn = Button(text='Почати', size_hint=(0.3, 0.4), pos_hint={'center_x': 0.5})
        self.btn.background_color = btn_color
        self.btn.on_press = self.next
        outer = BoxLayout(orientation='vertical', padding=8, spacing=8)
        outer.add_widget(instr)
        #outer.add_widget(lbl1)
        outer.add_widget(self.lbl_sec)
        outer.add_widget(line)
        outer.add_widget(self.btn)
        self.add_widget(outer)

    def sec_finished(self, *args):
        self.next_screen = True
        self.in_result.set_disabled(False)
        self.btn.set_disabled(False)
        self.btn.text = 'Продовжити'

    def next(self):
        if not self.next_screen:
            self.btn.set_disabled(True)
            self.lbl_sec.start()
        else:
            global p1
            p1 = check_int(self.in_result.text)
            if p1 == False or p1 <= 0:
                p1 = 0
                self.in_result.text = str(p1)
            else:
                self.manager.current = 'sits'
class CheckSits(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.next_screen = False

        instr = Label(text=txt_sits, size_hint=(0.5, 1))
        self.lbl_sits = Sits(30)
        self.run = Runner(total=30, steptime=1.5, size_hint=(0.4, 1))
        self.run.bind(finished=self.run_finished)

        line = BoxLayout()
        vlay = BoxLayout(orientation='vertical', size_hint=(0.3, 1))
        vlay.add_widget(self.lbl_sits)
        line.add_widget(instr)
        line.add_widget(vlay)
        line.add_widget(self.run)
```

```
        self.btn = Button(text='Почати', size_hint=(0.3, 0.2), pos_hint={'center_x': 0.5})
        self.btn.background_color = btn_color
        self.btn.on_press = self.next

        outer = BoxLayout(orientation='vertical', padding=8, spacing=8)
        outer.add_widget(line)
        outer.add_widget(self.btn)

        self.add_widget(outer)

    def run_finished(self, instance, value):
        self.btn.set_disabled(False)
        self.btn.text = 'Продовжити'
        self.next_screen = True

    def next(self):
        if not self.next_screen:
            self.btn.set_disabled(True)
            self.run.start()
            self.run.bind(value=self.lbl_sits.next)
        else:
            self.manager.current = 'pulse2'

class PulseScr2(Screen):
    def __init__(self, **kwargs):
        self.next_screen = False

        self.stage = 0
        super().__init__(**kwargs)
        instr = Label(text=txt_test3)
        line1 = BoxLayout(size_hint=(0.8, None), height='30sp')
        self.lbl_sec = Seconds(15)
        self.lbl_sec.bind(done=self.sec_finished)
        self.lbl1 = Label(text='Рахуйте пульс')

        lbl_result1 = Label(text='Результат:', halign='right')
        self.in_result1 = TextInput(text='0', multiline=False)
        line1.add_widget(lbl_result1)
        line1.add_widget(self.in_result1)
        line2 = BoxLayout(size_hint=(0.8, None), height='30sp')
        lbl_result2 = Label(text='Результат після відпочинку:', halign='right')
        self.in_result2 = TextInput(text='0', multiline=False)

        self.in_result1.set_disabled(True)
        self.in_result2.set_disabled(True)
        line2.add_widget(lbl_result2)
        line2.add_widget(self.in_result2)
        self.btn = Button(text='Почати', size_hint=(0.3, 0.5), pos_hint={'center_x': 0.5})
        self.btn.background_color = btn_color
        self.btn.on_press = self.next
```

```
        outer = BoxLayout(orientation='vertical', padding=8, spacing=8)
        outer.add_widget(instr)
        outer.add_widget(self.lbl1)
        outer.add_widget(self.lbl_sec)
        outer.add_widget(line1)
        outer.add_widget(line2)
        outer.add_widget(self.btn)
        self.add_widget(outer)

    def sec_finished(self, *args):
        if self.lbl_sec.done == True:
            if self.stage == 0:
                # закінчили перший підрахунок, відпочиваємо
                self.stage = 1
                self.lbl1.text = 'Відпочивайте'
                self.lbl_sec.restart(30)
                self.in_result1.set_disabled(False)
            elif self.stage == 1:
                # закінчили відпочинок, вважаємо
                self.stage = 2
                self.lbl1.text= 'Рахуйте пульс'
                self.lbl_sec.restart(15)
            elif self.stage == 2:
                self.in_result2.set_disabled(False)
                self.btn.set_disabled(False)
                self.btn.text = 'Завершити'
                self.next_screen = True
    def next(self):
        if not self.next_screen:
            self.btn.set_disabled(True)
            self.lbl_sec.start()
        else:
            global p2, p3
            p2 = check_int(self.in_result1.text)
            p3 = check_int(self.in_result2.text)
            if p2 == False:
                p2 = 0
                self.in_result1.text = str(p2)
            elif p3 == False:
                p3 = 0
                self.in_result2.text = str(p3)
            else:
                # переходимо
                self.manager.current = 'result'
class Result(Screen):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.outer = BoxLayout(orientation='vertical', padding=8, spacing=8)
        self.instr = Label(text = '')
        self.outer.add_widget(self.instr)
```

```
        self.add_widget(self.outer)
        self.on_enter = self.before
    def before(self):
        global name
        self.instr.text = name + '\n' + test(p1, p2, p3, age)
class HeartCheck(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(InstrScr(name='instr'))
        sm.add_widget(PulseScr(name='pulse1'))
        sm.add_widget(CheckSits(name='sits'))
        sm.add_widget(PulseScr2(name='pulse2'))
        sm.add_widget(Result(name='result'))
        return sm
app = HeartCheck()
app.run()
```

Додаток 3. Файл ruffier.py:

```
''' Модуль для розрахунку результатів проби Руф'є.

Сума вимірювань пульсу у трьох спробах (до навантаження, одразу після та після короткого відпочинку)
в ідеалі має бути не більше 200 ударів на хвилину.
Ми пропонуємо дітям вимірювати свій пульс протягом 15 секунд,
і наводимо результат до ударів за хвилину множенням на 4:
   S = 4* (P1 + P2 + P3)
Що далі цей результат від ідеальних 200 ударів, то гірше.
Традиційно таблиці даються для величини, поділеної на 10.

Індекс Руф'є
  IR = (S - 200) / 10
оцінюється за таблицею відповідно до віку:
      7-8          9-10          11-12          13-14          15+ (тільки для підлітків!)
чудов.   6.4 і менше   4.9 і менше      3.4 і менше      1.9 і менше      0.4 і менше
доб.   6.5 - 11.9    5 - 10.4       3.5 - 8.9       2 - 7.4          0.5 - 5.9
задов.  12 - 16.9     10.5 - 15.4     9 - 13.9        7.5 - 12.4       6 - 10.9
слабкий 17 - 20.9     15.5 - 19.4     14 - 17.9       12.5 - 16.4      11 - 14.9
незад.  21 і більше   19.5 і більше   18 і більше     16.5 і більше    15 і більше

для всіх вікових груп результат "незад." відрізняється від "слабкий" на 4,
той від "задовільного" на 5, а "добрий" від "задов." - на 5.5

тому напишемо функцію ruffier_result(r_index, level), яка отримуватиме
розрахований індекс Руф'є та рівень "невуд" для віку тестованого, і віддавати результат
'''
```

```
# тут задаються рядки, за допомогою яких викладено результат:
txt_index = "Ваш індекс Руф’є: "
txt_workheart = "Працездатність серця: "
txt_nodata = "'немає даних для такого віку"'
txt_res = []
txt_res.append("'низька.Терміново зверніться до лікаря!"')
txt_res.append("'задовільна.Зверніться до лікаря!"')
txt_res.append("'середня. Можливо, варто додатково обстежитись у лікаря."')
txt_res.append("'вище середнього"')
txt_res.append("'висока"')

def ruffier_index(P1, P2, P3):
    "' повертає значення індексу за трьома показниками пульсу для звірки з таблицею"'
    return (4 * (P1+P2+P3) - 200) / 10

def neud_level(age):
    "' варіанти з віком менше 7 і дорослим треба обробляти окремо,
        тут підбираємо рівень "незадовільно" тільки всередині таблиці:
        у віці 7 років "неуд" - це індекс 21, далі кожні 2 роки він знижується на 1.5 до значення 15 в 15-16
років "'
    norm_age = (min(age, 15) - 7) // 2  # кожні 2 роки різниці від 7 років перетворюються на одиницю - аж
до 15 років
    result = 21 - norm_age * 1.5 # множимо кожні 2 роки різниці на 1.5, так розподілені рівні у таблиці
    return result

def ruffier_result(r_index, level):
    "'функція отримує індекс Руф’є і інтерпретує його,
        повертає рівень готовності: число від 0 до 4
        (що вище рівень готовності, то краще).
    "'
    if r_index >= level:
        return 0
    level = level - 4 # це не буде виконуватися, якщо ми вже повернули відповідь "незадовільно"
    if r_index >= level:
        return 1
    level = level - 5 # аналогічно, потрапляємо сюди, якщо рівень як мінімум "задов."
    if r_index >= level:
        return 2
    level = level - 5.5 # наступний рівень
    if r_index >= level:
        return 3
    return 4 # тут виявилися, якщо індекс менше від усіх проміжних рівнів, тобто крут, що тестується.

def test(P1, P2, P3, age):
    "' цю функцію можна використовувати зовні модуля для підрахунків індексу Руф’є.
        Повертає готові тексти, які залишається намалювати у потрібному місці
        Використовує для текстів константи, задані на початку цього модуля."'
    if age < 7:
        return (txt_index + "0", txt_nodata) # таємниця ця не для цього тесту
    else:
```

```
    ruff_index = ruffier_index(P1, P2, P3) # розрахунок
    result = txt_res[ruffier_result(ruff_index, neud_level(age))] # інтерпретація, переведення числового рівня підготовки до текстових даних
    res = txt_index + str(ruff_index) + '\n' + txt_workheart + result
    return res
```

Додаток 4. Файл runner.py:

```
from kivy.properties import NumericProperty, BooleanProperty
from kivy.uix.button import Button
from kivy.animation import Animation

from kivy.uix.boxlayout import BoxLayout

class Runner(BoxLayout):
    value = NumericProperty(0) # скільки зроблено переміщень
    finished = BooleanProperty(False) # чи зроблені всі переміщення

    def __init__(self,
                 total=10,  steptime=1, autorepeat=True,
                 bcolor=(0.90, 0.15, 0.15, 1),
                 btext_inprogress='Присідання',
                 **kwargs):

        super().__init__(**kwargs)

        self.total = total
        self.autorepeat = autorepeat
        self.btext_inprogress = btext_inprogress
        self.animation = (Animation(pos_hint={'top': 0.1}, duration=steptime/2)
                          + Animation(pos_hint={'top': 1.0}, duration=steptime/2))
        self.animation.on_progress = self.next
        self.btn = Button(size_hint=(1, 0.1), pos_hint={'top': 1.0}, background_color=bcolor)
        self.add_widget(self.btn)

    '''def restart(self, total):
        self.total = total
        self.start()'''

    def start(self):
        self.value = 0
        self.finished = False
        self.btn.text = self.btext_inprogress
        if self.autorepeat:
            self.animation.repeat = True
        self.animation.start(self.btn)

    def next(self, widget, step):
```

```
        if step == 1.0:
            self.value += 1
            if self.value >= self.total:
                self.animation.repeat = False
                self.finished = True
```

Додаток 5. Файл seconds.py:

```
from kivy.uix.label import Label
from kivy.clock import Clock
from kivy.properties import BooleanProperty

class Seconds(Label):
    done = BooleanProperty(False)

    def __init__(self, total, **kwargs):
        self.done = False
        self.current = 0
        self.total = total
        my_text = "Пройшло секунд: " + str(self.current)
        super().__init__(text=my_text)

    def restart(self, total, **kwargs):
        self.done = False
        self.total = total
        self.current = 0
        self.text = "Пройшло секунд: " + str(self.current)
        self.start()

    def start(self):
        Clock.schedule_interval(self.change, 1)

    def change(self, dt):
        self.current += 1
        self.text = "Пройшло секунд: " + str(self.current)
        if self.current >= self.total:
            self.done = True
            return False
```

Додаток 6. Файл sits.py:

```
from kivy.uix.label import Label
from kivy.clock import Clock

class Sits(Label):
    def __init__(self, total, **kwargs):
        self.current = 0
        self.total = total
```

```
    my_text = "Залишилось присідань: " + str(self.total)
    super().__init__(text=my_text, **kwargs)

def next(self, *args):
    self.current += 1
    remain = max(0, self.total - self.current)
    my_text = "Залишилось присідань: " + str(remain)
    self.text=my_text
```