

5.3.2. Interfejs graficzny (GUI)

Interfejs graficzny stworzono w oparciu o rozszerzenie do modułu tkinter – CustomTkinter. Rozszerzenie zostało przygotowane przez Tom’a Schimansky’ego, który to zamieścił pełną dokumentację na platformie GitHub. Moduł zaimplementowano przy użyciu funkcji *import customtkinter*. CustomTkinter zapewnia nowocześniejszy wygląd widżetów w stosunku do klasycznej biblioteki Tkinter. Rozszerzenie umożliwia użytkownikowi wybór jednego z dwóch motywów, ciemnego i jasnego. W programie „System PPOŻ” zastosowano ciemny motyw, a kolor domyślnego podświetlenia widżetów takich jak przyciski zadeklarowano jako niebieski.

```
#set appearance mode to dark
customtkinter.set_appearance_mode("dark")
#set theme for widget - buttons etc
customtkinter.set_default_color_theme("blue")
```

Ustawienie motywu interfejsu i widżetów

Wymiary wyświetlanego programu to 850px szerokości na 600px wysokości. Wyświetlane okno nazwano *self.window* i powiązано z oknem głównym o standardowej nazwie *root*. Oknu nadano nazwę „System PPOŻ” i dodano ikonę programu widoczną w lewym górnym rogu. Obraz w formacie ico został zaimportowany z dysku użytkownika. W celu ustanowienia bardziej ustrukturyzowanego kodu program został zbudowany w oparciu o klasę główną *CountDown*, która pełni funkcję nadrzędną w stosunku do pozostałych funkcji *def*.

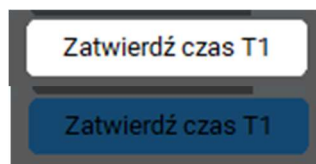
```
class CountDown(customtkinter.CTk):
def __init__(self, root):
super().__init__()
self.window = root
self.window.geometry(f'{850}x{600}')
self.window.title('System PPOŻ')
self.window.iconbitmap('C:/Users/milos/Automatyzacja-projekt/fire_image.ico')
self.window.configure(bg='gray35')
```

Utworzenie *class*,
zadeklarowania
konstruktora klasy
__init__, uruchamiany
zawsze podczas
tworzenia nowego
obiektu, *self* jest
wartością umowną,
odnoszącą się do
samej siebie jak 'this'
w C++

Sposób deklaracji *CTkButton*, *CTkLabel*, *CTkFrame* i *CTkRadioButton* został przedstawiony poniżej. Format zapisu wszystkich widżetów znajduje się w dokumentacji rozszerzenia CustomTkinter - załącznik [X]. Label to widżet umożliwiający wyświetlenie w ramce o określonych wymiarach tekstu lub obrazu w formacie np. png lub jpg.

```
set_button = customtkinter.CTkButton(self.window,
text='Zatwierdź czas T1', fg_color="White", text_color="Black",
border_color="Black", command=self.Get_Time)
set_button.grid(row=3, column=2)
```

Ustawienie stylu przycisku, tekstu
(*text_font=(...)*), wyznaczenie
lokalizacji przycisku w oknie
aplikacji (*label.grid(...)*) oraz
powiązanie przycisku z funkcją
(*command=...*)



Rys. 33. Widok przycisku w programie System PPOŻ przed najechaniem kursorem myszy na przycisk i po

```
time_label = customtkinter.CTkLabel(self.window,
text="Ustaw czas T1", corner_radius=8,
text_font=("times new roman", 16, "bold"),
text_color="White", fg_color=("gray35"))
time_label.grid(row=0, column=2)
```

Ustawienie stylu Label, tekstu, wyznaczenie lokalizacji Label w oknie aplikacji

```
self.time_frame = customtkinter.CTkFrame(self.window,
corner_radius=10,width=1, height=1, bg='gray35').grid(row=4, column=2)
```

Ustawienie stylu Frame, tekstu, wyznaczenie lokalizacji Frame w oknie aplikacji

```
radiobutton_1=customtkinter.CTkRadioButton(master=self.window,
text="Alarmowanie 1 stopniowe",
command=self.radiobutton, variable=radio_var, value=1)
radiobutton_1.grid(row=11, column=6)
```

Ustawienie stylu RadioButton, tekstu, wyznaczenie lokalizacji Frame w oknie aplikacji



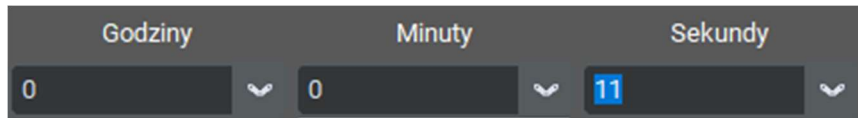
Rys. 34. Widok przycisku RadioButton w programie System PPOŻ przed najechaniem kursorem myszy na przycisk i po

Elementem charakterystycznym dla każdego z widżetów jest przedrostek `customtkinter.CTk(Nazwa widżetu)`, niezbędny do zaimplementowania rozszerzenia `CustomTkinter`. Umieszczenie widżetów w oknie aplikacji umożliwia `NAZWA.place()`, `NAZWA.pack()` lub `NAZWA.grid()`. W przypadku `.place()` widżet umieszczamy względem osi `x` i `y` np. `minuteEntry.place(x=60, y=40)`. Dla `.pack()` określamy stronę `side=TOP/BOTTOM/RIGHT` umiejscowienia w oknie aplikacji oraz `ipadx/y`, `padx/y` – iloma pikselami należy wypełnić widżet, w poziomie i w pionie. W finalnej wersji programu użyto `.grid()` określając rozmieszczenie widżetu względem wierszy i kolumn.

Czas T1 i T2 użytkownik deklaruje poprzez wybranie z widżetu `CTkComboBox` jednej z zaprogramowanych wartości czasu. Wartość czasu możemy wpisać z klawiatury lub wybrać z listy rozwijanej.

```
self.hour_combobox = customtkinter.CTkComboBox(master=self.window,
values=['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14',
'15', '16', '17', '18', '19', '20', '21', '22', '23', '24'],
command=combobox_callback)
self.hour_combobox.grid(row=2, column=1)
```

Wartości deklarujemy w
values=[1,2...], wybrana
wartość zostaje użyta w
funkcji
combobox_callback



Rys. 35. Widok widżetu ComboBox w programie System PPOŻ

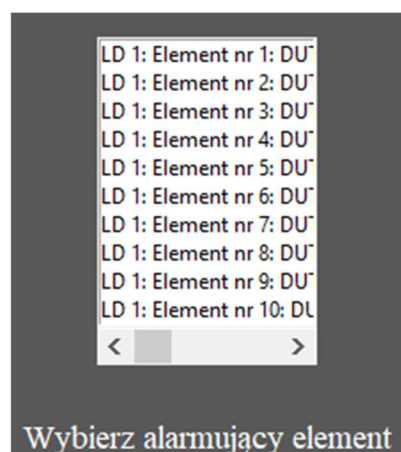
Listę elementów liniowych przedstawiono w tabelach nr 4,5 i 6, program System PPOŻ zawiera podgląd listy elementów i możliwość wyboru jednego z nich. Program odnosi się do pliku *Arkusz Konfiguracyjny_6000_wer_7.075.xlsm* na dysku użytkownika, a dokładniej do Arkusza „Tabela Linii dozorowych”, z którego pobiera wartości w zadeklarowanych komórkach np. $f\{ws[\"B3\"].value\}: \{ws[\"C3\"].value\}: \{ws[\"D3\"].value\}'$.

```
wb=load_workbook('C:\\Users\\milos\\Automatyzacja-projekt\\ArkuszKonfiguracyjny_6000_wer_7.075.xlsm')
ws = wb[\"Tabela Linii dozorowych\"]
my_listbox.insert(END, f{ws[\"B3\"].value}: {ws[\"C3\"].value}: {ws[\"D3\"].value}')
```

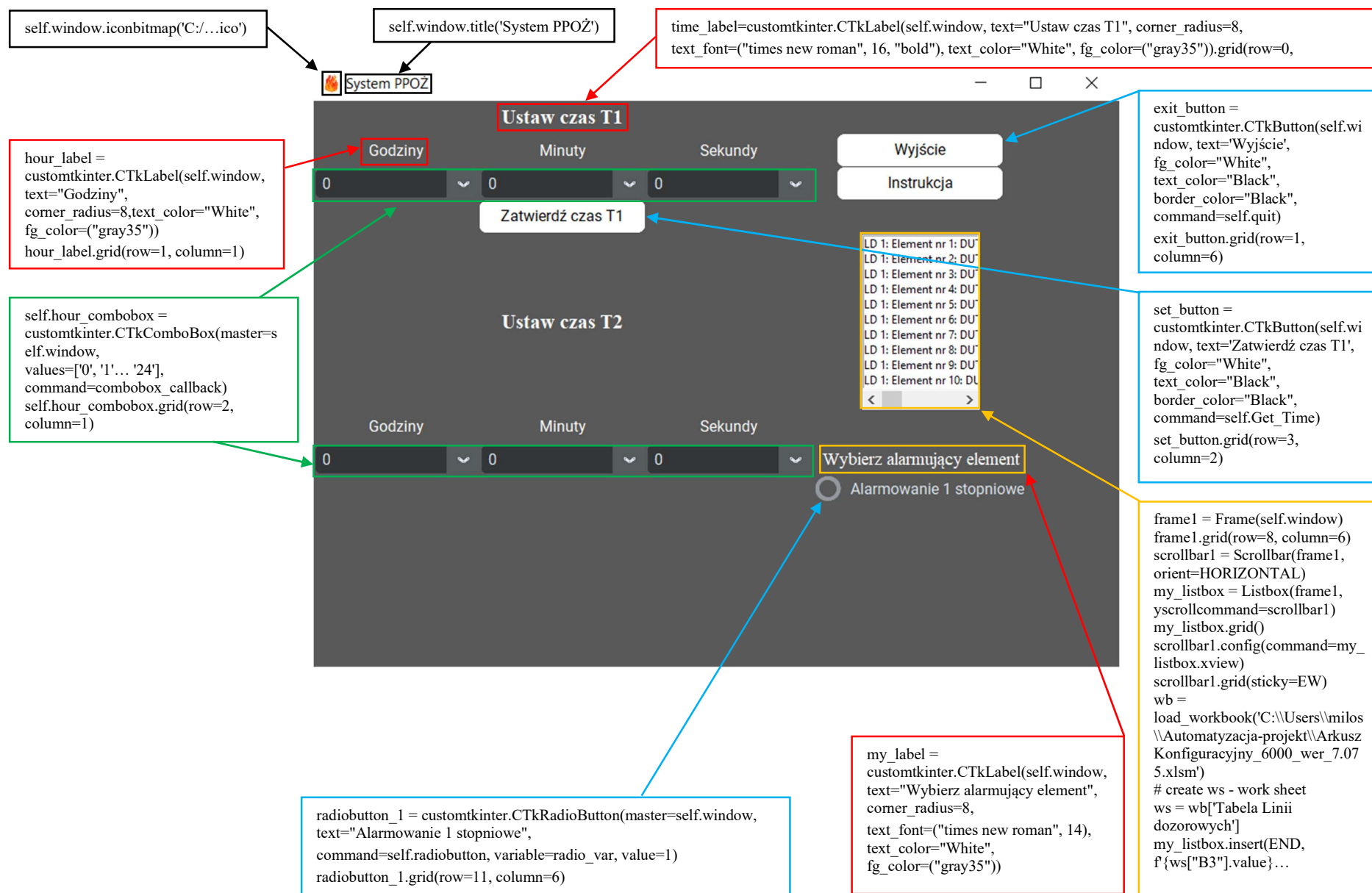
Lista *my_listbox* została umieszczona w *frame1*, dodatkowo utworzono *scrollbar1* w orientacji horyzontalnej, służącą do przewijania i podglądu danych.

```
frame1 = Frame(self.window)
frame1.grid(row=8, column=6)
scrollbar1 = Scrollbar(frame1, orient=HORIZONTAL)
scrollbar1.config(command=my_listbox.xview)
scrollbar1.grid(sticky=EW)
```

scrollbar1 odnosi się do funkcji
my_listbox.xview odpowiadającej za
wyświetlanie wybranego elementu z listy
my_listbox po kliknięciu lewym
przyciskiem myszy



Rys. 36. Widok widżetów *my_listbox*, *scrollbar1* i *frame1* w programie System PPOŻ



Rys. 37. Interfejs graficzny programu System PPOŻ (GUI)

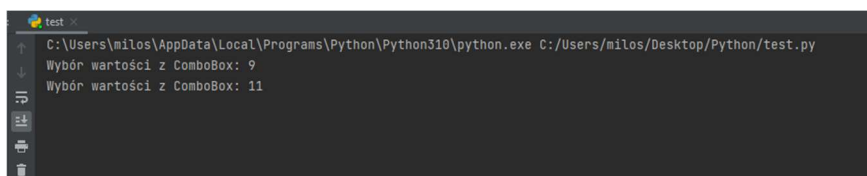
5.3.3. Funkcje realizowane przez program

W języku programowania Python funkcja opisano jako *def* to grupa powiązanych ze sobą instrukcji, które wykonują określone zadanie. Podział programu na funkcje sprawia, że kod źródłowy jest bardziej zorganizowany i łatwiejszy w zarządzaniu. Widżety takie jak przyciski (*button*) odwołują się do określonej funkcji poprzez *command=self.root.NAZWA_FUNKCJI*, co sprawia, że po kliknięciu lewym przyciskiem myszy w przycisk wykonywane są instrukcje zawarte w funkcji *def*.

```
def combobox_callback(choice):
```

```
    print("Wybór wartości z ComboBox:", choice)
```

Wybranie wartości z listy
hour/minute/second_combobox powoduje jej
wyświetlenie w oknie Powershell, wierszu
poleceń lub używanego przez użytkownika
środowiska programistycznego



Rys. 38. Wyświetlenie wybranej wartości z widżetu Combobox

```
def quit(self):  
    root.destroy()
```

Funkcja *def quit()* jest powiązana z przyciskiem „Wyjście”,
kliknięcie przycisku powoduje zamknięcie programu System PPOŻ

```
def select(self):  
    my_label.config(text=my_listbox.get(ANCHOR))
```

Funkcja *def select()* umożliwia wybór
wartości z *frame1*

5.3.4. Timer

Program System PPOŻ zawiera kilka funkcji odpowiedzialnych za obsługę Timer’a. Najistotniejsza z nich to *Get_Time*, która zostaje uruchomiona po wciśnięciu przycisku „Zatwierdź czas T1” wyświetla niedostępne wcześniej przyciski – *start_button* („Wywołanie alarmu”), *pause_button* („Potwierdzenie”), *ROP_button* („ROP”), *cancel_button* („Kasowanie”).

```
start_button = customtkinter.CTkButton(self.window,  
text='Wywołanie alarmu', fg_color="White",  
text_color="Black", border_color="Black", command=self.Threading)  
start_button.grid(row=5, column=2)
```

Zadeklarowanie przycisku
start_button („Wywołanie
alarmu”)

```
pause_button = customtkinter.CTkButton(self.window,  
text='Potwierdzenie', fg_color="Green",  
text_color="White", border_color="Black", command=self.pause_time)  
pause_button.grid(row=6, column=1)
```

Zadeklarowanie przycisku
pause_button („Potwierdzenie”)

```
ROP_button = customtkinter.CTkButton(self.window,
text='ROP', fg_color="White",
text_color="Red", border_color="Black",command=self.ROP)
ROP_button.grid(row=6, column=3)
```

Zadeklarowanie przycisku
ROP_button („ROP”)

```
cancel_button = customtkinter.CTkButton(self.window,
text='Kasowanie',fg_color="Red",
text_color="White", border_color="Black", command=self.Cancel)
cancel_button.grid(row=6, column=2)
```

Zadeklarowanie przycisku
cancel_button („Kasowanie”)



Rys. 39. Wyświetlenie przycisków po aktywowaniu funkcji Get_Time

Get_Time pobiera wartości z widżetów hour_label, minute_label oraz second_label a następnie przelicza każdą z wartości na sekundy.

```
def Get_Time(self):
```

```
    self.time_display = Label(self.time_frame,
                              font=('Helvetica', 16, "bold"),
                              bg='gray35', fg='yellow')
    self.time_display.grid(row=4, column=2)
```

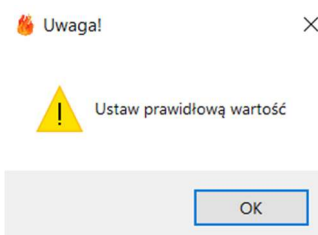
Funkcja Get_Time wyświetla pozostały czas alarmowania po wprowadzeniu wartości i zatwierdzeniu odliczania przyciskiem start_button („Wywołanie alarmu”)

```
    try:
```

```
        h = (int(self.hour_combobox.get()) * 3600)
        m = (int(self.minute_combobox.get()) * 60)
        s = (int(self.second_combobox.get()))
        self.time_left = h + m + s
```

Przeliczenia wartości zawartych w widżetach hour_combobox, minute_combobox, second_combobox na sekundy, a następnie zsumowanie tych wartości, czego wynikiem jest pozostały czas alarmowania

W przypadku nie ustawienia przez użytkownika wartości w jednym z widżetów (wartość standardowa w każdym Combobox to 0) zostanie wyświetlony komunikat o błędzie.



Rys. 40. Komunikat o błędzie w przypadku nie ustawienia wartości innej niż "0"

```
except (RuntimeError, TypeError, NameError):
    pass
```

Wcześniej zastosowana instrukcja *try* może zawierać jeden lub więcej instrukcji *Exception*. Najpierw funkcja *Get_Time* realizuje instrukcję *try*, jeżeli wystąpi jeden z wymienionych warunków *RuntimeError* itd. komunikat o błędzie nie jest wyświetlany za sprawą instrukcji *pass*

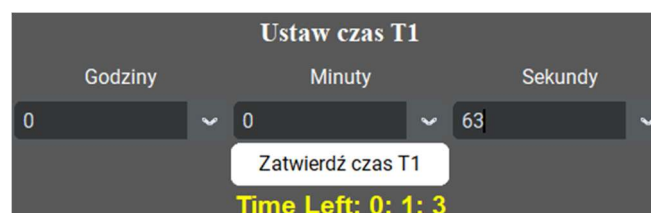
Funkcja *start_time* jest powiązana z przyciskiem *start_button* („Wywołanie alarmu”) i odpowiada za rozpoczęcia odliczania czasu zadeklarowanego przez użytkownika w widżetach *Combobox*.

```
def start_time(self):
    self.pause = False
    while self.time_left > 0:
        mins, secs = divmod(self.time_left, 60)
        hours = 0
        if mins > 60:
            hours, mins = divmod(mins, 60)
```

Podczas wykonywania funkcji *start_time* instrukcja *pause* jest zanegowana. Pętla *while* wykonuje instrukcję tak długo jak warunek jest spełniony.

Funkcja *divmod(dividend, divisor)* składa się z dwóch wartości:
dividend – liczba, którą chcemy podzielić,
divisor – liczba, przez którą chcemy podzielić.
 Funkcja *divmod* przelicza wartość w widżetach *Combobox*, jeśli użytkownik wprowadzi liczbę większą niż 60

Użytkownik wprowadzając wartość do widżetu np. *self.second_combobox* większą niż 60 i zatwierdzając przyciskami *set_button*, a następnie *start_button* inicjuje wykonanie funkcji *divmod*, która dzieli wartość przez 60 i zwraca resztę z dzielenia. Na rysunku nr zaprezentowano działanie funkcji.



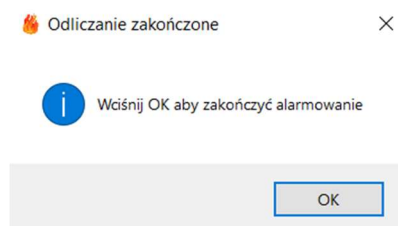
Rys. 41. Działanie funkcji *divmod*

```
self.time_display.config(
    text=f"Time Left: {hours}: {mins}: {secs}")
self.time_display.update()
time.sleep(1)
self.time_left = self.time_left - 1
if self.time_left <= 0:
    process = multiprocessing.Process(target=playsound,
                                     args=('C:/Users/milos/Automatyzacja-projekt/wariant1.mp3',))
    process.start()
    self.time_display.config(text=f"Time Left: {0}: {0}: {0}")
```

Wyświetla i aktualizuje czas wpisany przez użytkownika w widżetach *ComboBox* po zatwierdzeniu przyciskami *set_button*, a następnie *start_button*

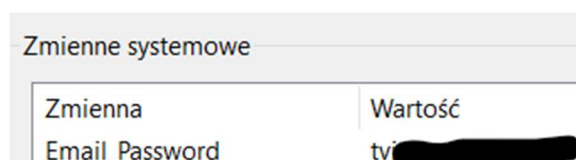
Po upływie czasu T1 i/lub T2 następuje uruchomienie sygnału dźwiękowego

`messagebox.showinfo('Odliczanie zakończone',
'Wciśnij OK aby zakończyć alarmowanie')` } Wyświetlenie alertu po upływie czasu T1
i/lub T2



Rys. 42. Alert wyświetlany po upływie czasu T1 i/lub T2

Kolejna sekcja funkcji *start_time* odpowiada za wysłanie na wybrany adres email (receiver) wiadomości z zadeklarowanego adresu (sender). Na potrzeby programu System PPOŻ stworzono nowy adres mailowy w serwisie gmail. Podczas tworzenia nowego adresu zezwolono na podwójną weryfikację konta, jedną z metod jest weryfikacja poprzez numer telefonu, natomiast drugie istotniejsza z punktu widzenia działania programu to utworzenia hasła logowania do konta dla aplikacji i urządzeń użytkownika, które domyślnie nie wspierają dwu stopniowej weryfikacji. Utworzone hasło można wkleić bezpośrednio do kodu źródłowego programu System PPOŻ, jednak udostępnianie kodu będzie automatycznie wiązało się z rozpowszechnianiem danych wrażliwych dla twórcy kodu. Zdecydowano się na utworzenie zmiennej środowiskowej w systemie windows, której wartość program pobierze, za sprawą instrukcji *os.environ.get('...')*. Można również użyć instrukcji *password = input(„Wprowadź hasło: „)*, gdzie użytkownik sam wprowadza hasło dostępu.



Rys. 43. Ustawienie zmiennej środowiskowej w systemie Windows

```
email_sender = 'arturnowak.test@gmail.com'
email_password = os.environ.get("Email_Password")
email_receiver = 'miłoszwojtko@gmail.com'
subject = "Uwaga Pożar!"
body = """"Uwaga! W budynku wykryto pożar!""""
email = EmailMessage()
email['From'] = email_sender
email['To'] = email_receiver
email['Subject'] = subject
```

} Zadeklarowanie adresata i nadawcy
maila, określenie tematu i
wiadomości wysyłanej drogą
mailową


```

email.set_content(body)
context = ssl.create_default_context()
with smtplib.SMTP_SSL('smtp.gmail.com', 465,
context=context) as smtp:
    smtp.login(email_sender, email_password)
    smtp.sendmail(email_sender, email_receiver,
email.as_string())
process.terminate()
self.Clear_Screen()
if self.pause == True:
    break

```

SMTP to lokalny serwer debugowania, korzystający z modułu smtpd, który jest preinstalowany z Pythonem. Uruchomienie lokalnego serwera sprawia, że nie trzeba zajmować się szyfrowaniem wiadomości ani używać poświadczeń do logowania się do serwera poczty e-mail.

Funkcja `start_time` zostaje przerwana jeśli dojdzie do zatrzymania czasu T1 i/lub T2



Rys. 44. Wiadomość e-mail wysyłana po spełnieniu warunku `self.time_left <= 0`

Funkcja *Cancel* jest powiązana z przyciskiem *cancel_button* („Kasowanie”) i odpowiada za zresetowanie Timer’a i widżetów *ComboBox* do wartości domyślnej – „0”. Wyświetlany jest zaktualizowany czas *time_display* - "Pozostały czas: {0}: {0}: {0}".

```

def Cancel(self):
    self.pause = True
    self.hour_combobox.set(0)
    self.minute_combobox.set(0)
    self.second_combobox.set(0)
    self.hour2_combobox.set(0)
    self.minute2_combobox.set(0)
    self.second2_combobox.set(0)
    self.time_display.config(text=f"Pozostały czas: {0}: {0}: {0}")
    self.time_display.update()

```

Zatrzymanie wykonywania programu poprzez instrukcję `self.pause = True`, zaaktualizowanie wartości w widżetach *ComboBox* oraz reset pozostałego czasu alarmowania do {0}: {0}: {0}

Funkcja *pause_time* jest powiązana z przyciskiem *pause_button* („Potwierdzenie”) i odpowiada za zatrzymanie Timer’a. Funkcja *divmod* przelicza wartość w widżetach *ComboBox*, pozostająca do zakończenia odliczenia i wyświetla czas poprzez instrukcję *time_display.config*.

```
def pause_time(self):
    self.pause = True
    mins, secs = divmod(self.time_left, 60)
    hours = 0
    if mins > 60:
        hours, mins = divmod(mins, 60)
    self.time_display.config(text=f"Pozostały czas: {hours}: {mins}: {secs}")
    self.time_display.update()
```

Zatrzymanie wykonywania programu poprzez instrukcję `self.pause = True`, zaaktualizowanie wartości w widżetach ComboBox oraz update pozostałego czasu alarmowania

Sposób działania przycisku *pause_button* zależy od wyboru

wariantu alarmowania. Standardowo program działa w wariantcie alarmowania II stopniowego (rys. 27), użycie przycisku *pause_button* powoduje pobranie wartości z widżetów *hour2_combobox*, *minute2_combobox* oraz *second2_combobox*. Wartości są przeliczane na sekundy i aktualizowane wraz z upływaniem czasu *time_display*. Uwaga! Ze względu na opóźnienie 2 sekund od momentu wciśnięcia przycisku *pause_button* do rozpoczęcia odliczania czasu T2, różnicę dodano do wyświetlanego czasu - $self.time_left = h + m + (s + 2)$.

```
h = (int(self.hour2_combobox.get()) * 3600)
m = (int(self.minute2_combobox.get()) * 60)
s = (int(self.second2_combobox.get()))
self.time_left = h + m + (s + 2)
time.sleep(1)
self.time_left = self.time_left - 1
self.x = Thread(target=self.start_time, daemon=True)
self.x.start()
```

Przeliczenie wartości w ComboBox na sekundy, zaktualizowanie pozostałego czasu. *Threading* służy do uruchamiania wielu wątków (zadań, instrukcji, funkcji) w tym samym czasie – w tym konkretnym przypadku funkcję *start_time* i *pause_time* mogą działać w tym samym czasie.

Jeśli użytkownik nie ustawi wartości czasu T2, funkcja pobierze wartość wejściową, czyli „0”. Spowoduje to uruchomienie alarmu II stopnia, zaraz po zakończeniu odliczania czasu T1, de facto program działać będzie w wariantcie alarmowania I stopniowego, mimo, że ten wariant alarmowania nie został potwierdzony wciśnięciem przycisku *radiobutton_1* („Alarmowanie I stopniowe”).

Zgodnie z zasadą działania algorytmu wciśnięcie przycisku *ROP_button* („ROP”) spowoduje natychmiastowe przejście programu w II stopień alarmowania, co skutkuje odtworzeniem sygnału dźwiękowego oraz wysłaniem wiadomości e-mail.

```
def ROP(self):
    self.pause = True
    self.hour_combobox.set(0)
    self.minute_combobox.set(0)
    self.second_combobox.set(0)
    self.hour2_combobox.set(0)
    self.minute2_combobox.set(0)
```

Zatrzymanie wykonywania programu poprzez instrukcję `self.pause = True`, zaktualizowanie wartości w widżetach ComboBox oraz update pozostałego czasu alarmowania

```

self.second2_combobox.set(0)
process = multiprocessing.Process(target=playsound,
                                args=('C:/Users/milos/Automatyzacja-projekt/wariant1.mp3',))
process.start()
self.time_display.config(text=f"Pozostały czas: {0}: {0}: {0}")
messagebox.showinfo('Odliczanie zakończone',
                    email_sender = 'arturnowak.test@gmail.com'
                    email_password = os.environ.get("Email_Password")
                    email_receiver = 'miloszwojtko@gmail.com'

                    subject = "Uwaga Pożar!"
                    body = """"Uwaga! W budynku wykryto pożar!""""
                    email = EmailMessage()
                    email['From'] = email_sender
                    email['To'] = email_receiver
                    email['Subject'] = subject
                    email.set_content(body)
                    context = ssl.create_default_context()
                    with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=context) as smtp:
                        smtp.login(email_sender, email_password)
                        smtp.sendmail(email_sender, email_receiver, email.as_string())
                    process.terminate()

```

Po upływie czasu T1 i/lub T2 następuje uruchomienie sygnału dźwiękowego

Zadeklarowanie adresata i nadawcy maila, określenie tematu i wiadomości wysyłanej drogą mailową

5.3.5. Radiobutton – wybór wariantu alarmowania

Przełączanie się między wariantem alarmowania I stopniowy, a II stopniowym jest możliwe dzięki widżetowi *radiobutton_1*. Po uruchomieniu program działa w wariantcie II stopniowym, gdyż taki zastosowano w projekcie systemu sygnalizacji pożarowej. Wybór I stopniowego wariantu alarmowania jest możliwy po wciśnięciu przycisku *radiobutton_1*. Przycisk uruchamia funkcję *radiobutton*, która blokuje *hour2_combobox*, *minute2_combobox*, *second2_combobox*, uniemożliwiając użytkownikowi zadeklarowanie czasu T2. Program wyświetla *radiobutton_2* za pomocą instrukcji *grid*. Przycisk *radiobutton_2* jest powiązany z funkcją *radiobutton2*, która restartuje program System PPOŻ.

```

def radiobutton(self):
    self.hour2_combobox = customtkinter.CTkComboBox(
        values=[""], state="readonly").grid(row=10, column=1)
    self.minute2_combobox = customtkinter.CTkComboBox(
        values=[""], state="readonly").grid(row=10, column=2)
    self.second2_combobox = customtkinter.CTkComboBox(
        values=[""], state="readonly").grid(row=10, column=3)
    radio_var = tkinter.IntVar(self)
    radiobutton_2 = customtkinter.CTkRadioButton(
        master=self.window, text="Alarmowanie 2 stopniowe",
        command=self.radiobutton2,
        variable=radio_var, value=2)
    radiobutton_2.grid(row=12, column=6)

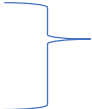
```

Instrukcja *state="readonly"* blokuje użytkownikowi zadeklarowanie wartości czasu T2

Funkcja *IntVar* zwraca wartość zmiennej jako liczbę całkowitą

Wyświetlenie widżetu *radiobutton_2*

```
def radiobutton2(self):  
    root.destroy()  
    os.startfile("test.py")
```



Funkcja *radiobutton2* restartuje program System PPOŻ