Raport aplikacji "Celebr"

Opis Ogólny

Aplikacja "Celebr" jest narzędziem do zarządzania osobistymi celami użytkownika. Umożliwia dodawanie, edycję, śledzenie postępu oraz usuwanie celów, które użytkownik chce osiągnąć w określonym czasie. Aplikacja została zaimplementowana w Pythonie, wykorzystując bibliotekę Tkinter do budowy interfejsu graficznego oraz bibliotekę PIL do obsługi obrazów.

Funkcje Aplikacji

1. Klasa MainMenu Klasa MainMenu odpowiada za główne okno aplikacji. Zawiera logo aplikacji, nazwę aplikacji oraz trzy przyciski: "Rozpocznij", "Autor" i "Wyjdź".

```
 class MainMenu:
    def __init__(self, master):
        # Inicjalizacja głównego okna
        self.master = master
        self.master.title("Celebr")
        self.master.geometry("1000x800")
        self.master.configure(background='#FFFFFF')

        # Logo aplikacji
        self.logo_image = Image.open("logo.png")
        self.logo_image = self.logo_image.resize((250, 250), Image.BILINEAR)
        self.logo_photo = ImageTk.PhotoImage(self.logo_image)
        self.logo_label = Label(self.master, image=self.logo_photo,
background='#FFFFFF')
        self.logo_label.pack()

        # Nagłówek aplikacji
        app_name_font = tkFont.Font(family="Avenir", size=50, weight="bold")
        self.app_name_label = Label(self.master, text="Celebr",
font=app_name_font, background='#FFFFFF', fg='#1a3b4c')
        self.app_name_label.pack()

        # Styl dla przycisków menu
        self.button_style = {
            'font': ('Avenir', 14),
            'background': '#1a3b4c',
            'foreground': '#FFFFFF',
            'activebackground': '#2c5d6b',
            'activeforeground': '#FFFFFF',
            'borderwidth': 2,
            'highlightthickness': 0,
            'padx': 10,
            'pady': 5,
        }
```

```
        # Przycisk "Rozpocznij"
        self.start_button = DepthButton(self.master, text="Rozpocznij",
command=self.show_goal_management_screen, **self.button_style)
        self.start_button.pack(fill='x', padx=10, pady=10)

        # Przycisk "Autor"
        self.author_button = DepthButton(self.master, text="Autor",
command=self.show_author_content, **self.button_style)
        self.author_button.pack(fill='x', padx=10, pady=10)

        # Przycisk "Wyjdź"
        self.quit_button = DepthButton(self.master, text="Wyjdź",
command=self.master.quit, **self.button_style)
        self.quit_button.pack(fill='x', padx=10, pady=(10, 40))

        # Lista celów
        self.goals = []  # Lista przechowująca cele

        # Wczytanie zapisanych celów (jeśli istnieją)
        self.load_goals()
```

2. Metoda show_goal_management_screen Metoda show_goal_management_screen tworzy okno do zarządzania celami. Zawiera listę celów, formularz do dodawania nowego celu oraz przyciski nawigacyjne.

```
    def show_goal_management_screen(self):
        # Zamknięcie głównego okna menu
        self.master.withdraw()

        # Tworzenie nowego okna Toplevel dla zarządzania celami
        goal_window = Toplevel(self.master)
        goal_window.title("Zarządzanie celami")
        goal_window.geometry("1000x800")

        # Ramka dla zawartości
        content_frame = Frame(goal_window, background='#FFFFFF')
        content_frame.pack(fill='both', expand=True, padx=20, pady=20)

        # Ramka z przyciskami celów po lewej stronie
        goals_frame = Frame(content_frame, background='#FFFFFF', width=300)
        goals_frame.pack(side='left', fill='y', padx=(0, 10), pady=10)

        Label(goals_frame, text="Twoje cele:", font=('Avenir', 18, 'bold'),
background='#FFFFFF', fg='#1a3b4c').pack(pady=10)

        # Lista przewijana dla przycisków celów
        self.goal_listbox = Listbox(goals_frame, selectmode='single',
background='#FFFFFF', exportselection=False)
        self.goal_listbox.pack(fill='both', expand=True, padx=10, pady=(0,10))

        self.goal_listbox.bind('<<ListboxSelect>>', self.on_goal_select)
```

```
        # Pobranie listy celów do wyświetlenia
        self.update_goals_list()

        # Formularz dodawania celu po prawej stronie
        self.add_goal_frame = Frame(content_frame, background='#FFFFFF',
highlightbackground='#1a3b4c', highlightthickness=2, padx=10, pady=10)
        self.add_goal_frame.pack(side='right', fill='both', expand=True, padx=20,
pady=20)

        Label(self.add_goal_frame, text="Dodaj nowy cel:", font=('Avenir', 18,
'bold'), background='#FFFFFF', fg='#1a3b4c').pack(pady=10)

        Label(self.add_goal_frame, text="Nazwa celu:",
background='#FFFFFF').pack(anchor='w', padx=10)
        self.goal_name_entry = Entry(self.add_goal_frame, font=('Avenir', 12),
relief="solid", bd=2)
        self.goal_name_entry.pack(fill='x', padx=10, pady=(0, 10))

        Label(self.add_goal_frame, text="Liczba dni:",
background='#FFFFFF').pack(anchor='w', padx=10)
        self.goal_days_entry = Entry(self.add_goal_frame, font=('Avenir', 12),
relief="solid", bd=2)
        self.goal_days_entry.pack(fill='x', padx=10, pady=(0, 10))

        add_button = DepthButton(self.add_goal_frame, text="Dodaj cel",
command=self.add_goal, **self.button_style)
        add_button.pack(fill='x', padx=10, pady=(10, 20))

        # Przycisk "Powrót do menu głównego"
        back_button = DepthButton(goals_frame, text="Powrót do menu głównego",
command=lambda: self.show_main_menu_and_close_window(goal_window),
**self.button_style)
        back_button.pack(side='bottom', fill='x', padx=10, pady=(20, 10))
```

3. Funkcje Pomocnicze add_goal: Dodaje nowy cel do listy celów. update_goals_list: Aktualizuje listę celów w oknie zarządzania celami. on_goal_select: Obsługuje zdarzenie wyboru celu z listy.

```
    def add_goal(self):
        goal_name = self.goal_name_entry.get()
        goal_days = self.goal_days_entry.get()

        if goal_name and goal_days.isdigit():
            goal_days = int(goal_days)
            new_goal = {
                'name': goal_name,
                'days': goal_days,
                'completed_days': [],
                'completed': False
            }
```

```
            self.goals.append(new_goal)
            self.save_goals()
            messagebox.showinfo("Dodano cel", f"Dodano nowy cel: {goal_name} na
{goal_days} dni.")
            self.goal_name_entry.delete(0, 'end')
            self.goal_days_entry.delete(0, 'end')
            self.update_goals_list()
        else:
            messagebox.showwarning("Błąd", "Proszę wprowadzić poprawną nazwę celu
i liczbę dni.")

    def update_goals_list(self):
        # Wyczyść listę celów przed aktualizacją
        self.goal_listbox.delete(0, END)

        # Dodaj cele do listy
        for i, goal in enumerate(self.goals, start=1):
            goal_name = goal['name']
            goal_days = goal['days']
            completed_days = goal['completed_days']
            days_left = goal_days - len(completed_days)

            if days_left <= 0:
                goal['completed'] = True
                self.goal_listbox.insert(END, f"Cel {i}: {goal_name} na
{goal_days} dni (Ukończony)")
                self.goal_listbox.itemconfig(END, {'fg': 'green'})
            else:
                self.goal_listbox.insert(END, f"Cel {i}: {goal_name} na
{goal_days} dni (Pozostało: {days_left} dni)")

    def on_goal_select(self, event):
        # Obsługa zdarzenia wyboru celu z listy
        try:
            selected_index = self.goal_listbox.curselection()[0]
            selected_goal = self.goals[selected_index]
            goal_name = selected_goal['name']
            goal_days = selected_goal['days']
            completed_days = selected_goal['completed_days']
            days_left = goal_days - len(completed_days)

            messagebox.showinfo("Szczegóły celu", f"Nazwa celu:
{goal_name}\nLiczba dni: {goal_days}\nDni pozostałe: {days_left}")
        except IndexError:
            pass

    def save_goals(self):
        # Zapis celów do pliku JSON
        with open('goals.json', 'w') as f:
            json.dump(self.goals, f, indent=4)

    def load_goals(self):
        # Wczytanie celów z pliku JSON (jeśli istnieje)
        if os.path.exists('goals.json'):
```

```
            with open('goals.json', 'r') as f:
                self.goals = json.load(f)
```

4. Klasa DepthButton Klasa DepthButton rozszerza standardowy przycisk Tkintera o efekty wizualne przy interakcjach takich jak najechanie myszką, naciśnięcie i puszczenie przycisku.

```
class DepthButton(Button):
    def __init__(self, master=None, **kwargs):
        super().__init__(master=master, **kwargs)
        self.config(
            relief="flat",
            borderwidth=0,
            highlightthickness=0,
            font=('Avenir', 14),
            background='#1a3b4c',
            foreground='#FFFFFF',
            activebackground='#2c5d6b',
            activeforeground='#FFFFFF'
        )

        self.bind("<Enter>", self.on_enter)
        self.bind("<Leave>", self.on_leave)

    def on_enter(self, e):
        self.config(
            relief="raised",
            borderwidth=2
        )

    def on_leave(self, e):
        self.config(
            relief="flat",
            borderwidth=0
        )
```

Technologie

Aplikacja wykorzystuje:

Python: Główny język programowania. Tkinter: Do tworzenia interfejsu graficznego. PIL (Pillow): Do manipulacji i wyświetlania obrazów.

Struktura Projektu

Projekt jest podzielony na:

gui.py: Główny plik programu, który inicjuje interfejs użytkownika. logo.png: Plik z logo aplikacji. goals.json: Plik przechowujący dane celów w formacie JSON.

Podsumowanie

Testowanie i debugowanie: Podczas rozwoju projektu napotkałem wiele błędów i nieprzewidzianych zachowań. Testowanie każdej nowej funkcji oraz debugowanie kodu było istotne w procesie rozwoju, aby upewnić się, że wszystko działa zgodnie z oczekiwaniami.

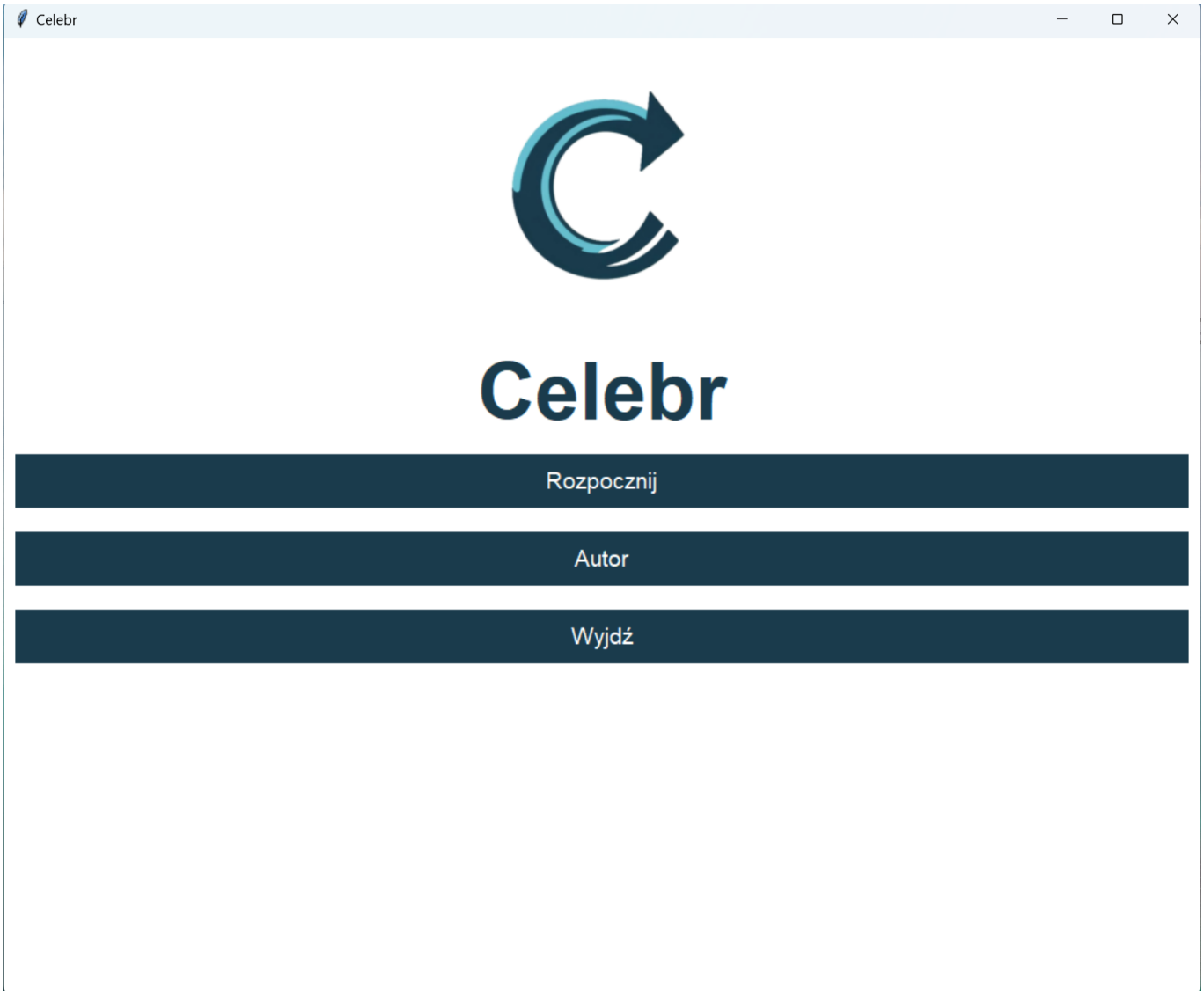
Iteracyjny proces: Projekt rozwijał się iteracyjnie, co oznaczało dodawanie, modyfikowanie i usuwanie funkcji w miarę potrzeb. Iteracyjny proces pozwolił na stopniowe ulepszanie kodu i dostosowywanie go do zmieniających się wymagań.

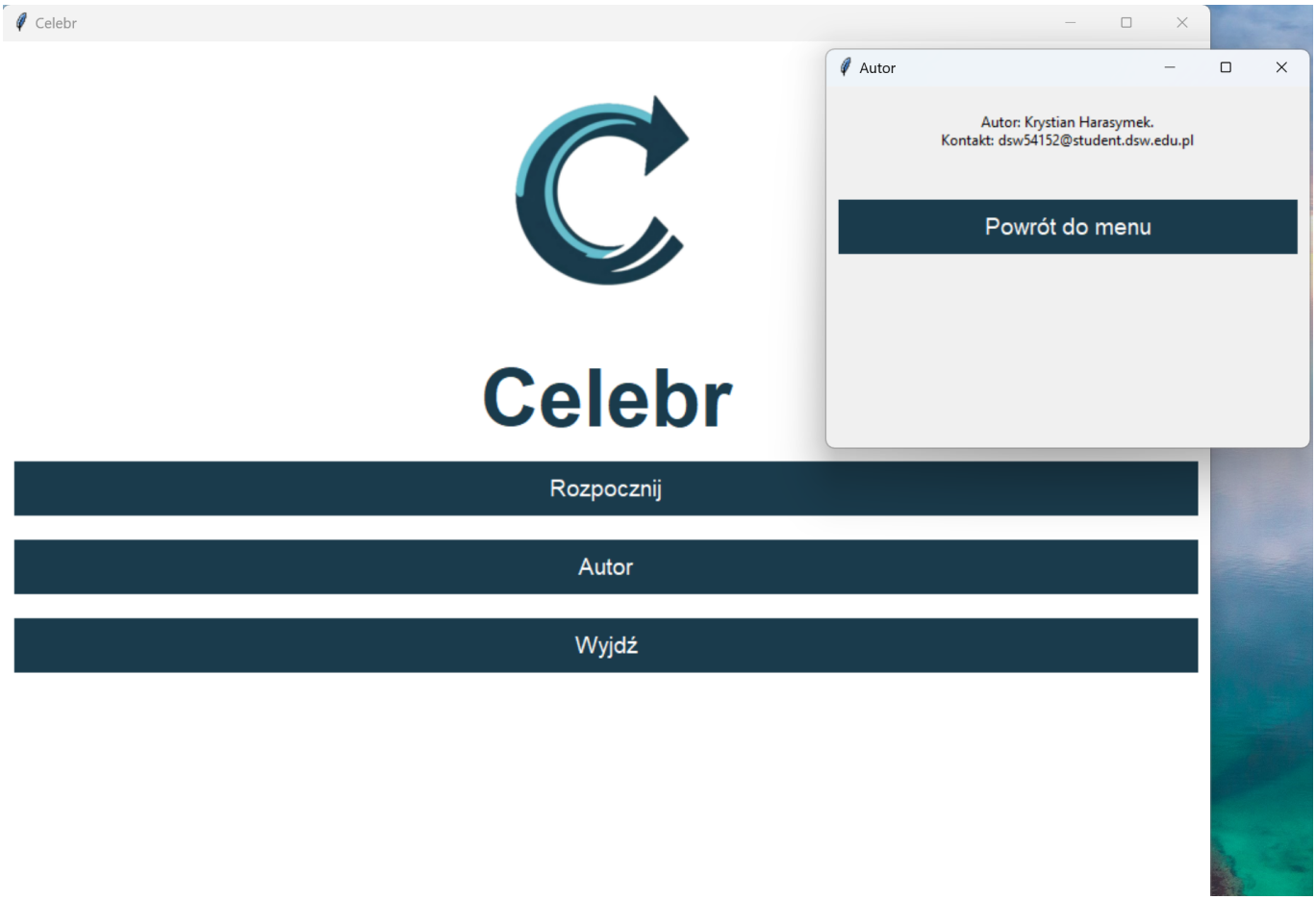
Zarządzanie błędami: W trakcie tworzenia projektu zrozumiałem, jak istotne jest skuteczne zarządzanie błędami. Dodanie obsługi wyjątków, walidacja danych wejściowych i odpowiednie komunikaty dla użytkownika pomogły w poprawnym działaniu programu i zapobieganiu "nieoczekiwanym" awariom.

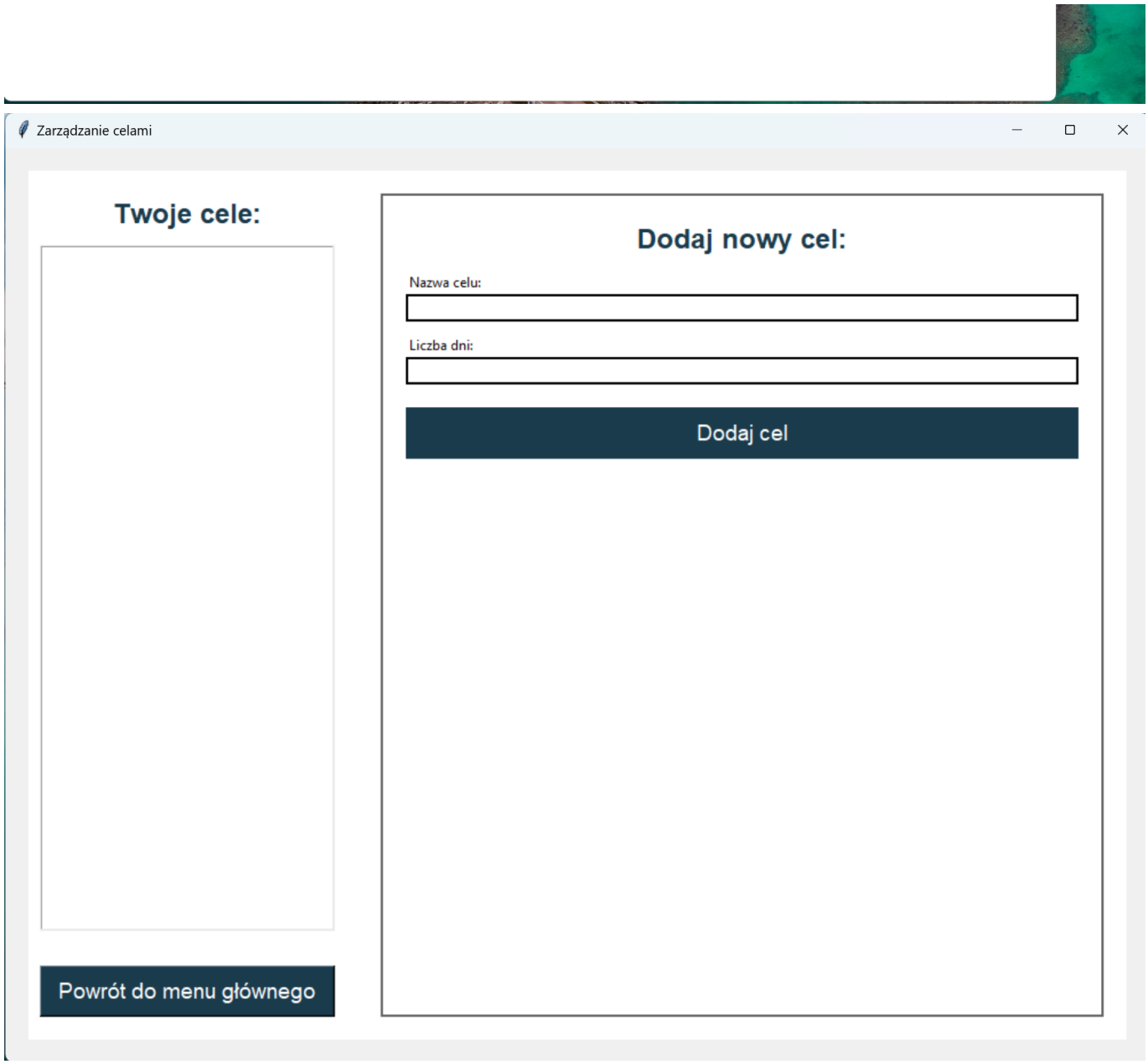
Organizacja kodu: Ważne jest utrzymanie czytelnego i dobrze zorganizowanego kodu. Dzięki odpowiedniemu nazewnictwu zmiennych i funkcji, komentarzom oraz logicznemu podziałowi kodu na moduły, było łatwiej zrozumieć i zarządzać projektem.

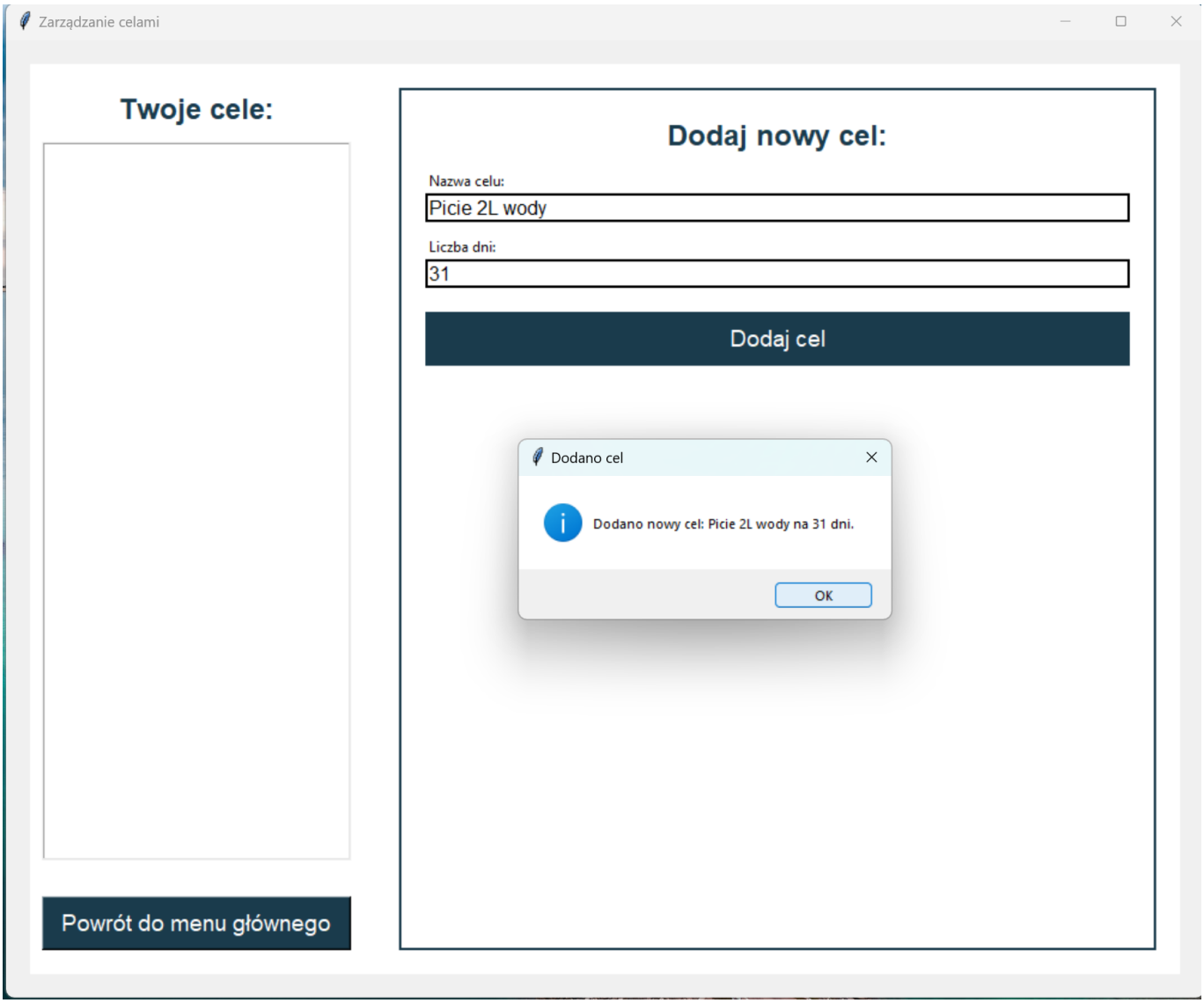
Kontrola wersji: Korzystanie z systemu kontroli wersji, takiego jak Git, okazało się niezwykle przydatne. Pozwoliło to na monitorowanie zmian w kodzie oraz przywracanie poprzednich wersji.

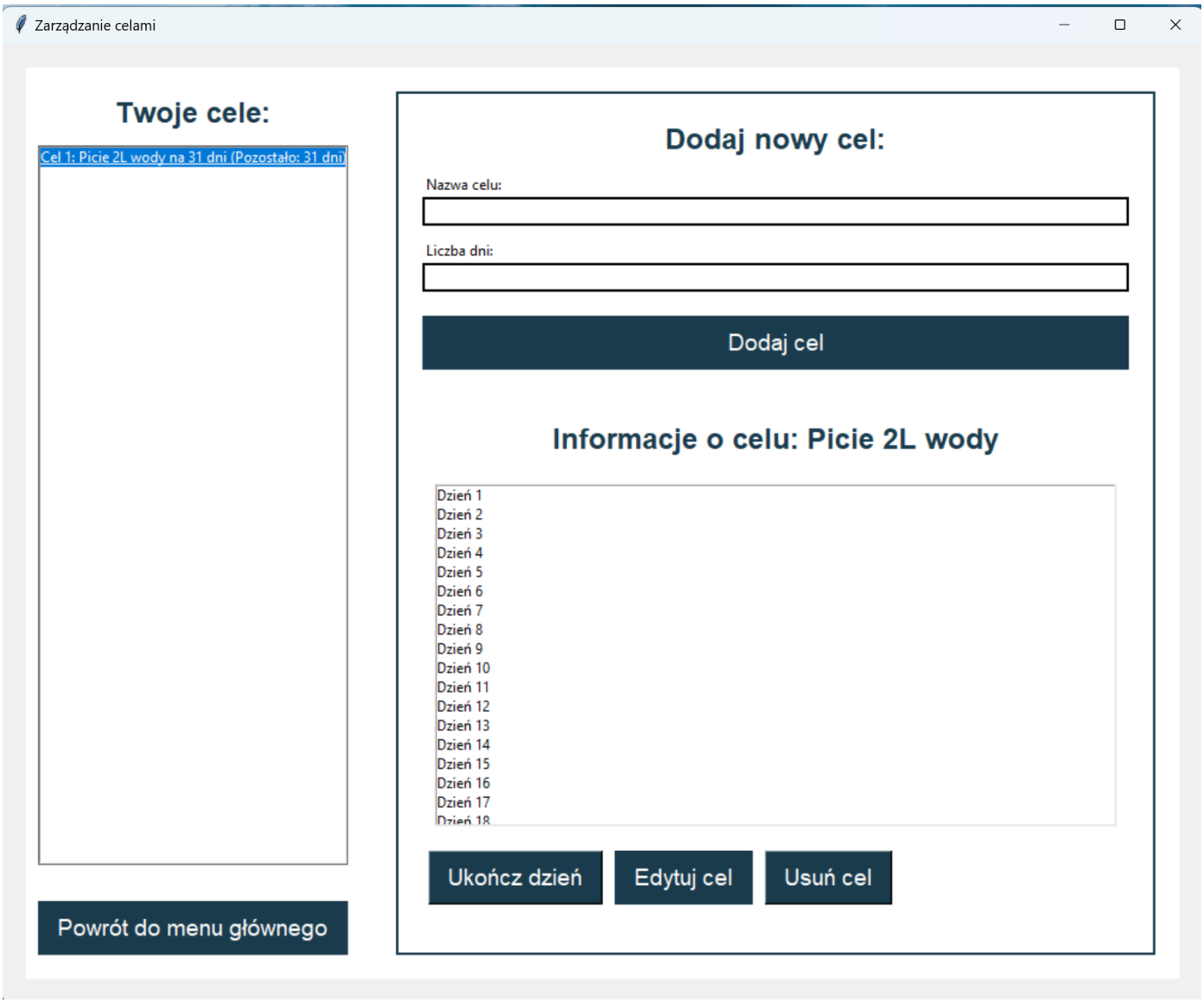
Zrzuty ekranu

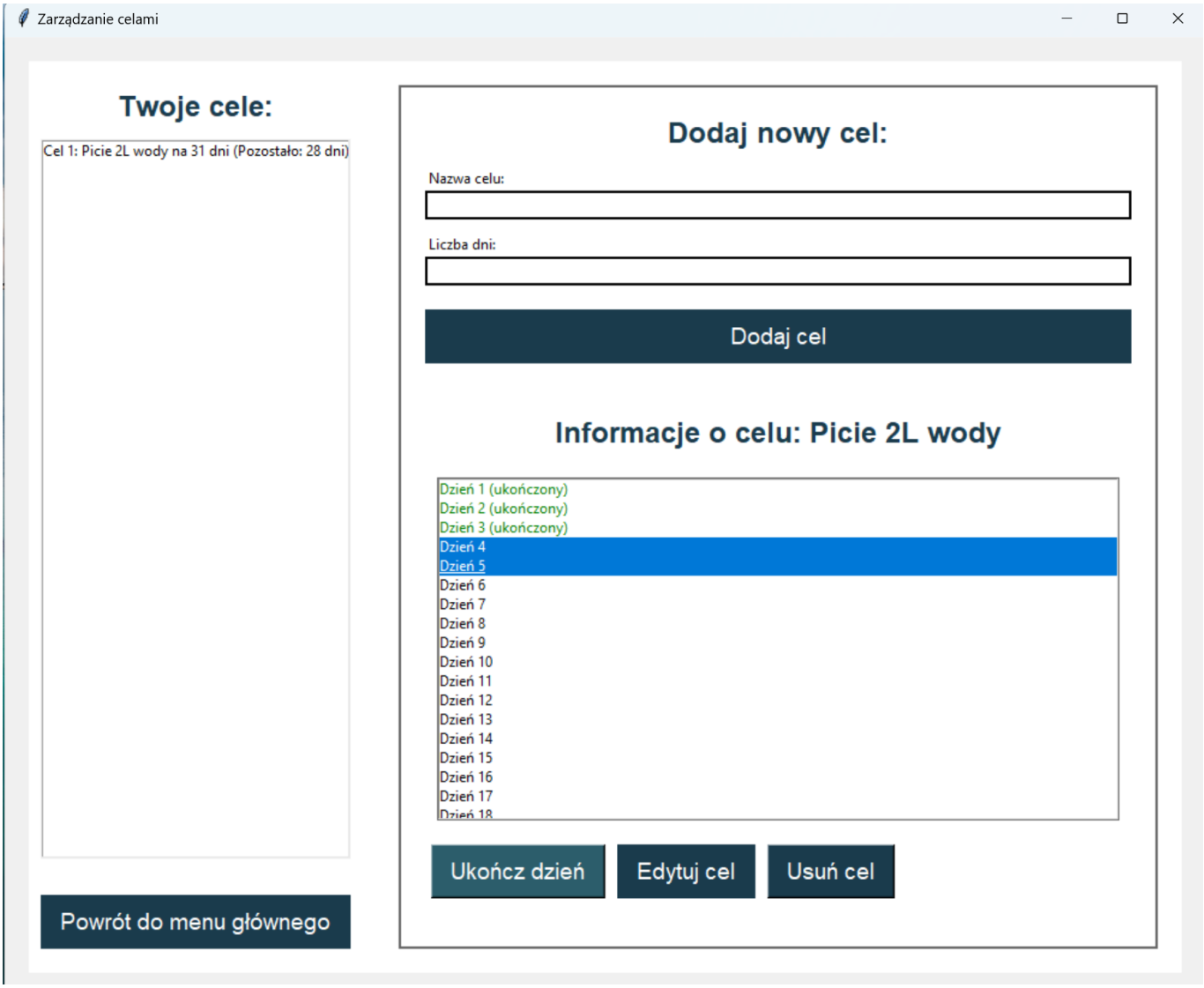

Celebr
Celebr
Rozpocznij
Autor
Wyjdź

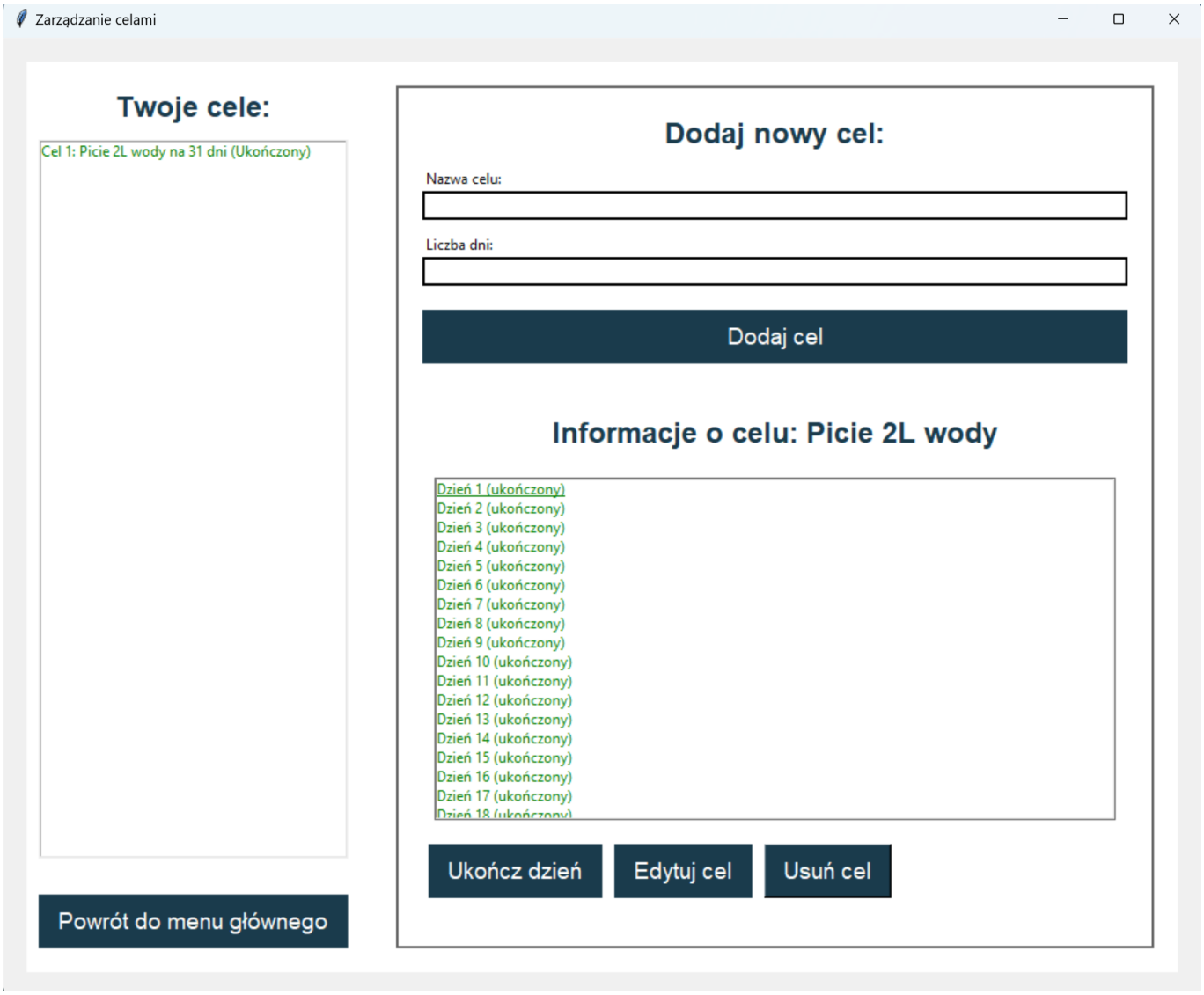

Celebr
Autor
Autor: Krystian Harasymek.
Kontakt: dsw54152@student.dsw.edu.pl
Powrót do menu
Celebr
Rozpocznij
Autor
Wyjdź

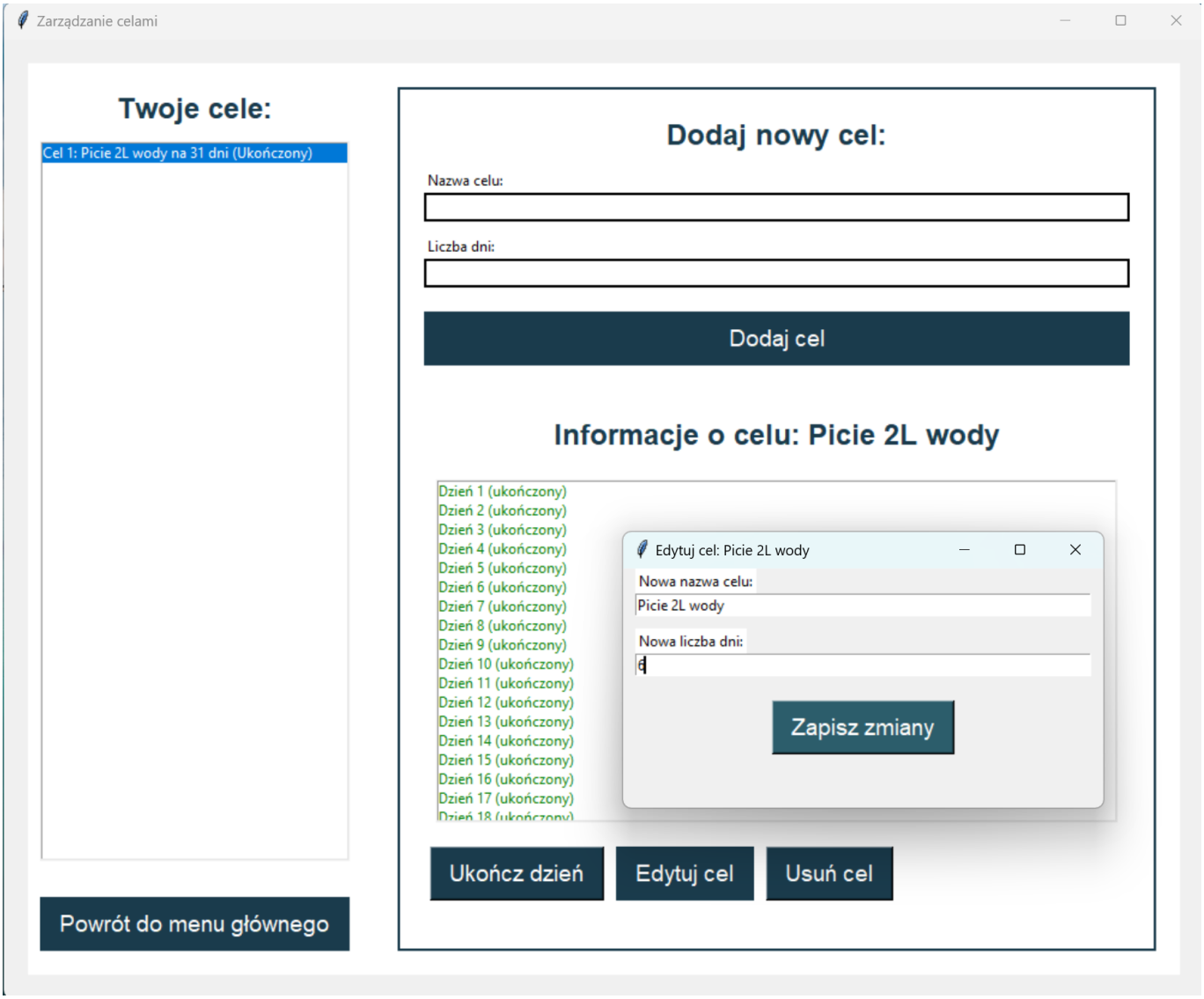

Zarządzanie celami
Twoje cele:
Dodaj nowy cel:
Nazwa celu:
Liczba dni:
Dodaj cel
Powrót do menu głównego


Zarządzanie celami
Twoje cele:
Powrót do menu głównego
Dodaj nowy cel:
Nazwa celu:
Picie 2L wody
Liczba dni:
31
Dodaj cel
Dodano cel
Dodano nowy cel: Picie 2L wody na 31 dni.
OK


Zarządzanie celami
Twoje cele:
Cel 1: Picie 2L wody na 31 dni (Pozostało: 31 dni)
Powrót do menu głównego
Dodaj nowy cel:
Nazwa celu:
Liczba dni:
Dodaj cel
Informacje o celu: Picie 2L wody
Dzień 1
Dzień 2
Dzień 3
Dzień 4
Dzień 5
Dzień 6
Dzień 7
Dzień 8
Dzień 9
Dzień 10
Dzień 11
Dzień 12
Dzień 13
Dzień 14
Dzień 15
Dzień 16
Dzień 17
Dzień 18
Ukończ dzień
Edytuj cel
Usuń cel


Zarządzanie celami
Twoje cele:
Cel 1: Picie 2L wody na 31 dni (Pozostało: 28 dni)
Powrót do menu głównego
Dodaj nowy cel:
Nazwa celu:
Liczba dni:
Dodaj cel
Informacje o celu: Picie 2L wody
Dzień 1 (ukończony)
Dzień 2 (ukończony)
Dzień 3 (ukończony)
Dzień 4
Dzień 5
Dzień 6
Dzień 7
Dzień 8
Dzień 9
Dzień 10
Dzień 11
Dzień 12
Dzień 13
Dzień 14
Dzień 15
Dzień 16
Dzień 17
Dzień 18
Ukończ dzień
Edytuj cel
Usuń cel


Zarządzanie celami
Twoje cele:
Cel 1: Picie 2L wody na 31 dni (Ukończony)
Powrót do menu głównego
Dodaj nowy cel:
Nazwa celu:
Liczba dni:
Dodaj cel
Informacje o celu: Picie 2L wody
Dzień 1 (ukończony)
Dzień 2 (ukończony)
Dzień 3 (ukończony)
Dzień 4 (ukończony)
Dzień 5 (ukończony)
Dzień 6 (ukończony)
Dzień 7 (ukończony)
Dzień 8 (ukończony)
Dzień 9 (ukończony)
Dzień 10 (ukończony)
Dzień 11 (ukończony)
Dzień 12 (ukończony)
Dzień 13 (ukończony)
Dzień 14 (ukończony)
Dzień 15 (ukończony)
Dzień 16 (ukończony)
Dzień 17 (ukończony)
Ukończ dzień
Edytuj cel
Usuń cel



Kody źródłowe

- [gui.py]

```
from tkinter import Tk, Frame, Label, Entry, Toplevel, messagebox, END, Listbox,
Button
from PIL import Image, ImageTk
import tkinter.font as tkFont
import json
import os

class DepthButton(Button):
    def __init__(self, parent=None, **kwargs):
        super().__init__(parent, **kwargs)
        self.default_bg = self["background"]
        self.bind("<Enter>", self.on_enter)
        self.bind("<Leave>", self.on_leave)
        self.bind("<ButtonPress-1>", self.on_press)
        self.bind("<ButtonRelease-1>", self.on_release)

    def on_enter(self, event):
        self["relief"] = "raised"
```

```
            self["background"] = "#2c5d6b"

        def on_leave(self, event):
            self["relief"] = "flat"
            self["background"] = self.default_bg

        def on_press(self, event):
            self["relief"] = "sunken"
            self["background"] = "#1a3b4c"

        def on_release(self, event):
            self["relief"] = "raised"
            self["background"] = "#2c5d6b"
            if self["command"]:
                self["command"]()

    class MainMenu:
        def __init__(self, master):
            self.master = master
            self.master.title("Celebr")
            self.master.geometry("1000x800")
            self.master.configure(background='#FFFFFF')

            # Logo aplikacji
            self.logo_image = Image.open("logo.png")
            self.logo_image = self.logo_image.resize((250, 250), Image.BILINEAR)
            self.logo_photo = ImageTk.PhotoImage(self.logo_image)

            # Logo na górze głównego okna
            self.logo_label = Label(self.master, image=self.logo_photo,
    background='#FFFFFF')
            self.logo_label.pack()

            # Nagłówek aplikacji
            app_name_font = tkFont.Font(family="Avenir", size=50, weight="bold")
            self.app_name_label = Label(self.master, text="Celebr",
    font=app_name_font, background='#FFFFFF', fg='#1a3b4c')
            self.app_name_label.pack()

            # Styl dla przycisków menu
            self.button_style = {
                'font': ('Avenir', 14),
                'background': '#1a3b4c',
                'foreground': '#FFFFFF',
                'activebackground': '#2c5d6b',
                'activeforeground': '#FFFFFF',
                'borderwidth': 2,
                'highlightthickness': 0,
                'padx': 10,
                'pady': 5,
            }

            # Przycisk "Rozpocznij"
            self.start_button = DepthButton(self.master, text="Rozpocznij",
```

```
command=self.show_goal_management_screen, **self.button_style)
        self.start_button.pack(fill='x', padx=10, pady=10)

        # Przycisk "Autor"
        self.author_button = DepthButton(self.master, text="Autor",
command=self.show_author_content, **self.button_style)
        self.author_button.pack(fill='x', padx=10, pady=10)

        # Przycisk "Wyjdź"
        self.quit_button = DepthButton(self.master, text="Wyjdź",
command=self.master.quit, **self.button_style)
        self.quit_button.pack(fill='x', padx=10, pady=(10, 40))

        # Lista celów
        self.goals = []  # Lista przechowująca cele

        # Wczytanie zapisanych celów (jeśli istnieją)
        self.load_goals()

    def show_goal_management_screen(self):
        # Zamknięcie głównego okna menu
        self.master.withdraw()

        # Tworzenie nowego okna Toplevel dla zarządzania celami
        goal_window = Toplevel(self.master)
        goal_window.title("Zarządzanie celami")
        goal_window.geometry("1000x800")

        # Ramka dla zawartości
        content_frame = Frame(goal_window, background='#FFFFFF')
        content_frame.pack(fill='both', expand=True, padx=20, pady=20)

        # Ramka z przyciskami celów po lewej stronie
        goals_frame = Frame(content_frame, background='#FFFFFF', width=300)
        goals_frame.pack(side='left', fill='y', padx=(0, 10), pady=10)

        Label(goals_frame, text="Twoje cele:", font=('Avenir', 18, 'bold'),
background='#FFFFFF', fg='#1a3b4c').pack(pady=10)

        # Lista przewijana dla przycisków celów
        self.goal_listbox = Listbox(goals_frame, selectmode='single',
background='#FFFFFF', exportselection=False)
        self.goal_listbox.pack(fill='both', expand=True, padx=10, pady=(0,10))

        self.goal_listbox.bind('<<ListboxSelect>>', self.on_goal_select)

        # Pobranie listy celów do wyświetlenia
        self.update_goals_list()

        # Formularz dodawania celu po prawej stronie
        self.add_goal_frame = Frame(content_frame, background='#FFFFFF',
highlightbackground='#1a3b4c', highlightthickness=2, padx=10, pady=10)
        self.add_goal_frame.pack(side='right', fill='both', expand=True, padx=20,
pady=20)
```

```
        Label(self.add_goal_frame, text="Dodaj nowy cel:", font=('Avenir', 18,
'bold'), background='#FFFFFF', fg='#1a3b4c').pack(pady=10)

        Label(self.add_goal_frame, text="Nazwa celu:",
background='#FFFFFF').pack(anchor='w', padx=10)
        self.goal_name_entry = Entry(self.add_goal_frame, font=('Avenir', 12),
relief="solid", bd=2)
        self.goal_name_entry.pack(fill='x', padx=10, pady=(0, 10))

        Label(self.add_goal_frame, text="Liczba dni:",
background='#FFFFFF').pack(anchor='w', padx=10)
        self.goal_days_entry = Entry(self.add_goal_frame, font=('Avenir', 12),
relief="solid", bd=2)
        self.goal_days_entry.pack(fill='x', padx=10, pady=(0, 10))

        add_button = DepthButton(self.add_goal_frame, text="Dodaj cel",
command=self.add_goal, **self.button_style)
        add_button.pack(fill='x', padx=10, pady=(10, 20))

        # Przycisk "Powrót do menu głównego"
        back_button = DepthButton(goals_frame, text="Powrót do menu głównego",
command=lambda: self.show_main_menu_and_close_window(goal_window),
**self.button_style)
        back_button.pack(side='bottom', fill='x', padx=10, pady=(20, 10))

    def add_goal(self):
        goal_name = self.goal_name_entry.get()
        goal_days = self.goal_days_entry.get()

        if goal_name and goal_days.isdigit():
            goal_days = int(goal_days)
            new_goal = {
                'name': goal_name,
                'days': goal_days,
                'completed_days': [],
                'completed': False
            }
            self.goals.append(new_goal)
            self.save_goals()
            messagebox.showinfo("Dodano cel", f"Dodano nowy cel: {goal_name} na
{goal_days} dni.")
            self.goal_name_entry.delete(0, 'end')
            self.goal_days_entry.delete(0, 'end')
            self.update_goals_list()
        else:
            messagebox.showwarning("Błąd", "Proszę wprowadzić poprawną nazwę celu
i liczbę dni.")

    def update_goals_list(self):
        # Wyczyść listę celów przed aktualizacją
        self.goal_listbox.delete(0, END)

        # Dodaj cele do listy
```

```
        for i, goal in enumerate(self.goals, start=1):
            goal_name = goal['name']
            goal_days = goal['days']
            completed_days = goal['completed_days']
            days_left = goal_days - len(completed_days)

            if days_left <= 0:
                goal['completed'] = True
                self.goal_listbox.insert(END, f"Cel {i}: {goal_name} na
{goal_days} dni (Ukończony)")
                self.goal_listbox.itemconfig(END, {'fg': 'green'})
            else:
                self.goal_listbox.insert(END, f"Cel {i}: {goal_name} na
{goal_days} dni (Pozostało: {days_left} dni)")

    def on_goal_select(self, event):
        # Obsługa zdarzenia wyboru celu z listy
        selected_indices = self.goal_listbox.curselection()
        if selected_indices:
            idx = selected_indices[0]
            self.show_goal_details(idx)

    def show_goal_details(self, idx):
        goal_name = self.goals[idx]['name']
        goal_days = self.goals[idx]['days']
        completed_days = self.goals[idx]['completed_days']

        # Usunięcie poprzedniej ramki z informacjami o celu, jeśli istnieje
        if hasattr(self, 'goal_details_frame'):
            self.goal_details_frame.destroy()

        # Utworzenie nowej ramki z informacjami o celu
        self.goal_details_frame = Frame(self.add_goal_frame, background='#FFFFFF')
        self.goal_details_frame.pack(fill='both', expand=True, padx=10, pady=10)

        Label(self.goal_details_frame, text=f"Informacje o celu: {goal_name}",
font=('Avenir', 18, 'bold'), background='#FFFFFF', fg='#1a3b4c').pack(pady=10)

        self.goal_days_listbox = Listbox(self.goal_details_frame,
selectmode='multiple', background='#FFFFFF')
        self.goal_days_listbox.pack(fill='both', expand=True, padx=10, pady=10)

        # Aktualizacja listy dni ukończonych
        self.update_goal_days_listbox(goal_days, completed_days)

        # Przycisk "Oznacz dzień ukończony"
        oznacz_dzien_button = DepthButton(self.goal_details_frame, text="Ukończ
dzień", command=lambda: self.oznacz_dzien_ukonczony(idx), **self.button_style)
        oznacz_dzien_button.pack(side='left', fill='x', padx=5, pady=(10, 20))

        # Przycisk "Edytuj cel"
        edit_button = DepthButton(self.goal_details_frame, text="Edytuj cel",
command=lambda: self.edit_goal(idx), **self.button_style)
        edit_button.pack(side='left', fill='x', padx=5, pady=(10, 20))
```

```
        # Przycisk "Usuń cel"
        remove_button = DepthButton(self.goal_details_frame, text="Usuń cel",
command=lambda: self.remove_goal(idx), **self.button_style)
        remove_button.pack(side='left', fill='x', padx=5, pady=(10, 20))

    def update_goal_days_listbox(self, goal_days, completed_days):
        self.goal_days_listbox.delete(0, END)
        for day in range(1, goal_days + 1):
            if day in completed_days:
                self.goal_days_listbox.insert(END, f"Dzień {day} (ukończony)")
                self.goal_days_listbox.itemconfig(END, {'fg': 'green'})
            else:
                self.goal_days_listbox.insert(END, f"Dzień {day}")

    def oznacz_dzien_ukonczony(self, idx):
        selected_indices = self.goal_days_listbox.curselection()
        selected_days = [int(self.goal_days_listbox.get(idx).split()[1]) for idx
in selected_indices]

        for day_number in selected_days:
            if day_number in self.goals[idx]['completed_days']:
                self.goals[idx]['completed_days'].remove(day_number)
            else:
                self.goals[idx]['completed_days'].append(day_number)

        self.save_goals()
        self.update_goal_days_listbox(self.goals[idx]['days'], self.goals[idx]
['completed_days'])
        self.update_goals_list()  # Dodajemy aktualizację listy celów po
zaktualizowaniu dni ukończonych
        self.check_goal_completion(idx)

    def check_goal_completion(self, idx):
        goal = self.goals[idx]
        if not goal['completed']:
            days_left = goal['days'] - len(goal['completed_days'])
            if days_left <= 0:
                goal['completed'] = True
                self.goal_listbox.delete(idx)
                self.goal_listbox.insert(idx, f"Cel {idx + 1}: {goal['name']} na
{goal['days']} dni (Ukończony)")
                self.goal_listbox.itemconfig(idx, {'fg': 'green'})
                messagebox.showinfo("Cel ukończony", f"Cel '{goal['name']}' został
ukończony!")

    def remove_goal(self, idx):
        del self.goals[idx]
        self.save_goals()
        self.update_goals_list()

    def edit_goal(self, idx):
        selected_goal = self.goals[idx]
```

```
        edit_window = Toplevel(self.master)
        edit_window.title(f"Edytuj cel: {selected_goal['name']}")
        edit_window.geometry("400x200")

        Label(edit_window, text="Nowa nazwa celu:",
background='#FFFFFF').pack(anchor='w', padx=10)
        new_name_entry = Entry(edit_window)
        new_name_entry.pack(fill='x', padx=10, pady=(0, 10))
        new_name_entry.insert(END, selected_goal['name'])

        Label(edit_window, text="Nowa liczba dni:",
background='#FFFFFF').pack(anchor='w', padx=10)
        new_days_entry = Entry(edit_window)
        new_days_entry.pack(fill='x', padx=10, pady=(0, 10))
        new_days_entry.insert(END, str(selected_goal['days']))

        save_button = DepthButton(edit_window, text="Zapisz zmiany",
command=lambda: self.save_edited_goal(idx, new_name_entry.get(),
new_days_entry.get()), **self.button_style)
        save_button.pack(pady=10)

    def save_edited_goal(self, idx, new_name, new_days):
        if new_name and new_days.isdigit():
            new_days = int(new_days)
            self.goals[idx]['name'] = new_name
            self.goals[idx]['days'] = new_days
            self.save_goals()
            self.update_goals_list()
            messagebox.showinfo("Zapisano zmiany", "Zaktualizowano dane celu.")
        else:
            messagebox.showwarning("Błąd", "Proszę wprowadzić poprawną nazwę celu
i liczbę dni.")

    def save_goals(self):
        # Zapisywanie celów do pliku JSON
        with open("goals.json", "w") as file:
            json.dump(self.goals, file, indent=4)

    def load_goals(self):
        # Ładowanie celów z pliku JSON (jeśli istnieją)
        if os.path.exists("goals.json"):
            with open("goals.json", "r") as file:
                self.goals = json.load(file)

    def show_main_menu_and_close_window(self, window):
        window.destroy()
        self.master.deiconify()

    def show_author_content(self):
        # Okno informacyjne dla ekranu Autor
        author_window = Toplevel(self.master)
        author_window.title("Autor")
        author_window.geometry("400x300")
```

```
        Label(author_window, text="Autor: Krystian Harasymek.\nKontakt:
dsw54152@student.dsw.edu.pl", padx=20, pady=20).pack()

        # Przycisk "Powrót do menu głównego"
        back_button = DepthButton(author_window, text="Powrót do menu",
command=author_window.destroy, **self.button_style)
        back_button.pack(fill='x', padx=10, pady=(20, 10))

if __name__ == "__main__":
    root = Tk()
    app = MainMenu(root)
    root.mainloop()
```

- [logo.png]
- [goals.json]

> Instrukcja przygotowana przez: [Krystian|54152] Data: [03.07.2024]