Projekt zawiera implementację generowania labiryntów z użyciem dwóch różnych algorytmów: Randomized Depth-First Search (DFS) oraz Randomized Kruskal's. Projekt obejmuje również funkcje do wizualizacji oraz menu dla użytkownika.

Randomized Depth-First Search (DFS):

- Algorytm DFS generuje labirynt poprzez rozpoczęcie od punktu startowego.
- Następnie, losowo wybiera jednego z sąsiadów i łączy się z nim, tworząc ścieżkę.
- Proces kontynuowany jest do momentu, gdy nie można już wybrać żadnego sąsiada, a więc następnie cofa się do poprzedniego punktu i kontynuuje poszukiwanie.
- W wyniku tego procesu powstaje losowo wygenerowany labirynt.

Randomized Kruskal's:

- Algorytm Kruskala generuje labirynt poprzez połączenie losowych komórek, tworząc sieć ścieżek.
- Początkowo, każda komórka labiryntu jest traktowana jako osobny zbiór (set).
- Krawędzie między komórkami labiryntu są tasowane w losowej kolejności.
- Algorytm iteruje po tasowanych krawędziach, próbując połączyć dwa sąsiadujące zbiory komórek.
- Jeśli krawędź łączy dwa różne zbiory komórek (zbiory nie są połączone), zostaje ona dodana do labiryntu jako ścieżka, a zbiory komórek są połączone.
- Ten proces jest kontynuowany aż do wyczerpania krawędzi lub utworzenia jednej spójnej składowej labiryntu.

Projekt składa się z kilku głównych komponentów:

Stałe: Stałe WALL, PATH, START, END służą do definiowania stanów komórek labiryntu.

Generacja Labiryntu: Funkcje generate_randomized_dfs_maze i generate_randomized_kruskal_maze generują labirynt za pomocą algorytmu DFS i Kruskala odpowiednio.

Wizualizacja Labiryntu: Funkcja generate_maze_image tworzy obraz reprezentujący labirynt w formie graficznej. Używa biblioteki PIL (Pillow) w tym celu.

Menu Użytkownika: Funkcje main_menu i main dostarczają interaktywne menu, które pozwala użytkownikowi wybrać opcję generacji labiryntu lub zakończenia programu.

Implementacja Algorytmu Depth-First Search (DFS) w Kodzie:

- Funkcja **generate_randomized_dfs_maze** implementuje algorytm DFS w kodzie.
- Na początku tworzona jest siatka labiryntu, gdzie wszystkie komórki są początkowo ustawione jako ściany (**WALL**).

- Wybierany jest losowy punkt startowy na siatce, który będzie punktem początkowym generacji labiryntu.
- Algorytm DFS rozpoczyna generowanie labiryntu od tego punktu startowego.
- Algorytm tworzy stos (**stack**), na którym śledzi aktualną ścieżkę w labiryncie. Początkowo, stos zawiera tylko punkt startowy.
- Proces generacji labiryntu polega na wybieraniu losowego sąsiada dla aktualnego punktu na szczycie stosu i łączeniu się z nim, tworząc ścieżkę. Losowość wyboru sąsiada zapewnia losowość labiryntu.
- Jeśli aktualny punkt ma sąsiada, który nie został jeszcze odwiedzony (tj. nie jest częścią aktualnej ścieżki), algorytm idzie do tego sąsiada i oznacza go jako część ścieżki.
- Jeśli aktualny punkt nie ma dostępnych sąsiadów lub wszyscy sąsiedzi są już częścią ścieżki, algorytm cofa się do poprzedniego punktu na stosie (co odpowiada cofaniu się w ścieżce).
- Proces kontynuowany jest do momentu, gdy na stosie nie ma więcej punktów do odwiedzenia (wszystkie dostępne ścieżki zostały zbadane).
- W wyniku tego procesu powstaje losowo wygenerowany labirynt, który jest reprezentowany jako dwuwymiarowa tablica, gdzie ściany (**WALL**) i ścieżki (**PATH**) tworzą strukturę labiryntu

Implementacja Algorytmu Kruskala w Kodzie:

- Funkcja **generate_randomized_kruskal_maze** implementuje algorytm Kruskala w kodzie.
- Na początku tworzona jest siatka labiryntu zbudowana ze ścian.
- Krawędzie między komórkami są reprezentowane jako elementy listy **walls**.
- Każda komórka labiryntu jest traktowana jako osobny zbiór i jest reprezentowana jako zbiór punktów **{(x, y)}**.
- Następnie algorytm tasuje krawędzie w liście **walls**.
- Iteruje po krawędziach, próbując połączyć zbiory komórek za pomocą funkcji **find_set** i **apply_union**.
- Gdy zbiory komórek są połączone, krawędź zostaje usunięta z listy **walls**, a ścieżka jest tworzona w labiryncie.
- Proces kontynuowany jest, aż wszystkie krawędzie zostaną przeanalizowane.

Instrukcja użytkowania:

- Po uruchomieniu programu pojawi się menu wyboru, które oferuje trzy opcje:
    1. Generuj i Zapisz Labirynt Randomized Depth-First Search
    2. Generuj i Zapisz Labirynt Randomized Kruskal's
    3. Wyjście z Programu

- Wybierz opcję generacji labiryntu, wpisując "1" lub "2", aby użyć odpowiedniego algorytmu, lub wybierz "3", aby wyjść z programu.
- Podaj szerokość i wysokość labiryntu (domyślnie ustawione jest na 10x10) lub pozostaw domyślne wartości, wciskając Enter.
- Podaj rozmiar komórki dla wizualizacji (domyślnie 20) lub pozostaw domyślną wartość, wciskając Enter.
- Labirynt zostanie wygenerowany i zapisany jako obraz w formacie png o odpowiedniej nazwie, zgodnie z wybranym algorytmem.
- Możesz również wybrać inną opcję z menu lub wyjść z programu.