Maze Generator

Dokumentacja projektu

Pola Dudek

3 stycznia 2025

Spis treści

1 Wstęp

1.1 Opis projektu

Maze Generator to program, którego głównym celem jest tworzenie labiryntów. Wykorzystuje on iteracyjną wersję algorytmu przeszukiwania w głąb (*DFS*), opartą na stosie. Użytkownik ma możliwość interaktywnego wprowadzenia wymiarów labiryntu, a sam proces generowania ścieżek wizualizowany jest w czasie rzeczywistym.

1.2 Specyfikacja

Program został zaimplementowany w języku Python i wykorzystuje następujące biblioteki:

- NumPy – dla operacji na tablicach
- Tkinter – w celu utworzenia interfejsu graficznego
- Random – dla losowości podczas generowania ścieżek labiryntu

2 Struktura programu

2.1 Klasa Stack

Implementuje stos wraz z podstawowymi operacjami:

```
class Stack:
    def __init__(self):
        self.elements = []
        self.top = -1

    def isEmpty(self):
        return self.top == -1

    def push(self, item):
        self.top += 1
        if self.top < len(self.elements):
            self.elements[self.top] = item
        else:
            self.elements.append(item)

    def pop(self):
        if self.isEmpty():
            raise IndexError("Stos jest pusty")
        item = self.elements[self.top]
        self.top -= 1
        return item

    def size(self):
        return self.top + 1

    def topEl(self):
        if not self.isEmpty():
            return self.elements[self.top]
        return None
```

Listing 1: klasa Stack

2.2 Klasa Maze

Odpowiada za inicjalizację planszy labiryntu oraz cały proces generowania kolejnych ścieżek. Klasa ta wewnętrznie koszysta z obiektu klasy `Stack`.

Metody:

- `generatingMaze(canvas, cell_size)` - jest to główna metoda odpowiedzialna za generowanie labiryntu, tworząc prześcia w odpowiednich komórkach.
- `isVisited(x, y)` - sprawdza, czy dana komórka została oznaczona jako odwiedzona.

- `makeVisited(x, y)` - oznacza daną komórkę jako odwiedzoną.
- `canBreakTheWall(x, y)` - sprawdza czy może i jeżeli tak, to usuwa ścianę pomiedzy dwiema komórkami
- `GUI(canvas, cell_size)` - odpowiada za graficzny interfejs programu

```
class Maze:
    def __init__(self, n, m):
        self.n = n
        self.m = m
        self.board = [[WALL] * (2*m+1) for _ in range(2*n+1)]
        self.visited = [[False] * m for _ in range(n)]
        self.stack = Stack()

    def generatingMaze(self, canvas, cell_size):
        directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
        startingPoint = (0, 0)
        self.stack.push(startingPoint)
        self.makeVisited(*startingPoint)

        startingX = 1
        startingY = 1
        self.canBreakTheWall(startingX, startingY)

        while not self.stack.isEmpty():
            currentPoint = self.stack.topEl()
            x, y = currentPoint

            neighbors = []
            for direction in directions:
                nextX, nextY = x + direction[0], y + direction[1]
                if 0 <= nextX < self.n and 0 <= nextY < self.m \
                   and not self.isVisited(nextX, nextY):
                    neighbors.append((nextX, nextY))

            if neighbors:
                nextPoint = random.choice(neighbors)
                nextX, nextY = nextPoint

                wallX = 2*x + 1 + (nextX - x)
                wallY = 2*y + 1 + (nextY - y)
                self.canBreakTheWall(wallX, wallY)
                self.canBreakTheWall(2 * nextX + 1, 2 * nextY + 1)

                self.makeVisited(nextX, nextY)
                self.stack.push(nextPoint)
            else:
                self.stack.pop()

            self.GUI(canvas, cell_size)

        self.board[1][0] = PASSAGE
        self.board[self.boardSizeX - 2][self.boardSizeY - 1] = PASSAGE
```

Listing 2: klasa Maze

3 Opis algorytmu

3.1 Ogólne założenia

- Labirynt reprezentowany jest jako dwuwymiarowa tablica, w której ściany zajmują tyle samo miejsca co przejścia, początkowo bez żadnych ścieżek. Wynika z tego potrzeba planszy o wymiarach (2n + 1) x (2m + 1), która reprezentuje realnie planszę n x m.
- Każda komórka odwiedzona może być tylko raz, zatem ścieżki nie mogą tworzyć cykli
- Ścieżka powstaje kiedy usunięta zostaje ściana pomiędzy dwiema komórkami.
- Labirynt ma wejście w lewym górnym roku oraz wyjście w prawym dolnym rogu.

3.2 Pseudokod

```
funkcja GenerujeLabirynt(n, m):
    labirynt := new int[2n+1][2m+1]
    odwiedzonePunkty := new bool[n][m]
    stos := new Stack()

    punktStartowy = (1, 1)
    stos.push(punktStartowy)
    dopoki stos nie jest pusty:
        obecnyElement = stos.topEl()
        sasiedzi = wszyscyNieodwiedzeniSasiedzi(obecnyElement)

        jezeli sasiedzi istnieja:
            nastepnaKomorka = random.choice(sasiedzi)
            usunSciane(obecnyElement, nastepnaKomorka)
            odwiedzonePunkty[nastepnaKomorka.x][nastepnaKomorka.y] = true
            stos.push(nastepnaKomorka)
        w przeciwnym razie:
            stos.pop()
```

Listing 3: Skrótowy pseudokod

4 Analiza Złożoności

4.1 Złożoność Czasowa

Algorytm DFS, dla generowania labiryntu charakteryzuje się złożonością czasową $O(n \times m)$, gdzie:

- n to liczba wierszy
- m to liczba kolumn

Ponieważ każda z komórek odwiedzana jest dokładnie raz, a wszystkie zaimplementowane operacje na stosie wykonują się w czasie $O(1)$.

4.2 Złożoność Pamięciowa

Całkowita złożoność pamięciowa również wynosi $O(n \times m)$:

- Tablica `visited`: $O(n \times m)$
- Tablica `board`: $O((2n+1) \times (2m+1)) = O(n \times m)$
- Stos `stack`: maksymalnie $O(n \times m)$ elementów

5 Uruchomienie oraz użytkowanie programu

5.1 Interfejs graficzny

GUI składa się z:

- Pól tekstowych przyjmujących rozmiary labiryntu
- Przycisku "Generuj labirynt"
- Obszaru wizualizacji procesu generowania, gdzie ściany reprezentowane są kolorem czarnym, przejścia białym oraz są odpowiednio ponumerowane, a wejście i wyjście są dodatkowo oznaczone.

5.2 Uruchomienie programu

1. Aby uruchomić program za pośrednictwem pliku Makefile, należy w folderze zawierającym plik ‘MazeGenerator.py‘ wprowadzić w terminalu następującą komendę: ‘make run‘. Spowoduje ona uruchomienie programu.
2. Aby określić parametry labiryntu, należy uzupełnić pola podpisane jako:
   - Liczba wierszy (n)
   - Liczba kolumn (m)
3. Następnie należy nacisnąć przycisk "Generuj labirynt", który rozpocznie tworzenie labiryntu.
4. Wizualizacja będzie przebiegała stopniowo, co pozwoli na obserwację działania zastosowanego algorytmu.

6 Podsumowanie

Maze Generator to program ilustrujący działanie algorytmu DFS w kontekście generowania labiryntów, z wykorzystaniem stosu.

Projekt posiada możliwość dalszej rozbudowy, na przykład poprzez implementację innych algorytmów generowania labiryntów.