Aufgabe 3: Zauberschule

Team-ID: 00879

Bearbeiter/-in dieser Aufgabe:
Karl Zschiebsch

12. November 2023

Inhaltsverzeichnis

Lösungsidee

Inspiriert wurde die Lösungsidee von Wegfindungsalgorithmen. Um den kürzesten Weg zu finden, markiert vom Startfeld ausgehend jedes Feld alle seine Nachbarn. Das Feld, welches zuerst das Zielfeld markiert, berechnet, von welchen Feldern es selbst markiert wurde. Der Weg, der darüber berechnet wurde, ist der kürzeste. Beachtet muss jedoch, dass nicht alle Felder “gleichzeitig” ihre Nachbarn markieren, sondern es eine Zeitverzögerung gibt.

Umsetzung

Um den kürzesten Weg zwischen A und B zu finden, wird zunächst A bestimmt. Von A aus werden alle angrenzenden Felder bestimmt (Das Feld im anderen Stockwerk zählt auch als angrenzend). Für jedes freie angrenzende Feld wird eine Aktion erstellt, die in drei Iterationen, falls das Stockwerk gewechselt wurde, sonst einer Iteration, abgearbeitet wird. Ein Feld gilt dann als frei, wenn es keine Wand ist und noch nicht markiert wurde. Wenn eine Aktion abgearbeitet wird, wird bestimmt, ob das Feld bereits markiert wurde. Wenn ein Feld noch nicht markiert wurde, markiert die Aktion das Feld und erstellt umgehend für alle verbleibenden freien Felder eine Aktion. Für die Iterationen wird ein Queue erstellt. Das Queue beinhaltet eine Reihe von Listen von Aktionen. Jede Liste stellt die Aktionen dar, die abgearbeitet werden sollen. Bei der Iteration wird die vorderste Liste mit Aktionen abgearbeitet und aus dem Queue entfernt. Dann rotieren die Elemente im Queue und an letzter stelle wird eine neue Liste hinzu gefügt.

Beispiele

Unten angefügt sind alle Ergebnisse für die jeweilige Dateien. Die Symbole, um den Weg darzustellen, sind die selben wie in der README.txt beschrieben. Vor dem Weg steht jeweils noch die Zeit, die man für den Weg brauchen würde.

```
zauberschule0.txt
(  8s) !>>!

zauberschule1.txt
(  4s) <<^^

zauberschule2.txt
( 14s) >>!>>!vv>>

zauberschule3.txt
( 28s) vv>>^^>>>>vv>>>>^^>>>>>>^^>>

zauberschule4.txt
( 84s) ^!^^^!<<<<<<^^<<<<<<<<<vv<<^^<<<<<<<<<!^^^^!<<!<<vv<<^^<<vv!
<<<<^^<<^^<<<<

zauberschule5.txt
(124s) !vv>>vv>>>>>>!>>>>>>>>>!vv>>>>>>>>>^^^^!^^>>!>>>>>>!
^^^^>>^^>>>>^^^^!^^>>^^>>>>>>!vv!vv>>>>>>>>!>>>>>>^^<<
```

Quellcode

Dies ist der Quellcode in Python. Es werden keine Bibliotheken benötigt.

```python
class Position:
    def __init__(self, x: int, y: int, z: int):
        self.x = x
        self.y = y
        self.z = z

class Building:
    def __init__(self, path: str):
        self.fields = [[], []]
        self.schedule: list[list['Action']] = [[], [], [], []]
        with open(path, 'r') as reader:
            self.n, self.m = [int(v) for v in reader.readline().split(' ')]
            for z in range(len(self.fields)):
                for y in range(self.n):
                    self.fields[z].append([])
                    for x in range(self.m):
                        c = Field(Position(x, y, z), reader.read(1))
                        if c.is_start():
                            self.start = c
                        self.fields[z][y].append(c)
                    reader.read(1)
                reader.read(1)
```

```
        if self.start is None:
            raise ValueError

    def is_inside(self, p: 'Position') -> bool:
        return 0 <= p.x < self.m and 0 <= p.y < self.n and 0 <= p.z <= 1

    def is_blocked(self, p: 'Position') -> bool:
        if self.is_inside(p):
            return self.get_field(p).is_blocked()
        return True

    def is_available(self, p: 'Position') -> bool:
        return self.is_inside(p) and self.get_field(p).is_available()

    def set_field(self, p: Position, val: 'Field') -> None:
        if self.is_inside(p):
            self.fields[p.x][p.y][p.z] = val
        else:
            raise IndexError(f"{p} is not in field!")

    def get_field(self, p: Position) -> 'Field':
        if self.is_inside(p):
            return self.fields[p.z][p.y][p.x]
        else:
            raise IndexError(f"{p} is not in field!")


class Field:
    def __init__(self, p: Position, t: str):
        if t == '\t':
            raise AssertionError(f"Invalid type for {p}")
        self.p = p
        self.type = t
        self.occupied = None

    def is_blocked(self) -> bool:
        return not self.is_floor() or self.occupied is not None

    def is_available(self) -> bool:
        return (self.is_floor() or self.is_end()) and self.occupied is None

    def is_wall(self) -> bool:
        return self.type == '#'

    def is_floor(self) -> bool:
        return self.type == '.'

    def is_start(self) -> bool:
        return self.type == 'A'

    def is_end(self) -> bool:
        return self.type == 'B'

    def neighbours(self, b: Building) -> list['Field']:
        neighbours = []
```

```
        for i in range(-1, 2, 2):
            p_0 = Position(self.p.x + i, self.p.y, self.p.z)
            if b.is_available(p_0):
                neighbours.append(b.get_field(p_0))
            p_1 = Position(self.p.x, self.p.y + i, self.p.z)
            if b.is_available(p_1):
                neighbours.append(b.get_field(p_1))
        return neighbours

    def __str__(self):
        return f"{self.p} => {self.type}"

    def __repr__(self):
        return f"Field({repr(self.p)}, '{self.type}')"


class Action:

    def __init__(self, target: Field, origin: 'Action' = None):
        self.target = target
        self.origin = origin

    def conquer(self, b: Building) -> None:
        t = self.target  # Shortcut
        for f in t.neighbours(b):
            b.schedule[1].append(Action(f, origin=self))
        pos = Position(t.p.x, t.p.y, (1, 0)[t.p.z])
        if b.is_available(pos):
            f = b.get_field(pos)
            b.schedule[3].append(Action(f, origin=self))

    def get_action(self) -> str:
        diff = self.target.p - self.origin.target.p
        if diff.x < 0:
            return '<'
        if diff.x > 0:
            return '>'
        if diff.y < 0:
            return '^'
        if diff.y > 0:
            return 'v'
        if diff.z != 0:
            return '!'

    def traceback(self) -> str:
        if self.target.is_start():
            return ''
        return self.origin.traceback() + self.get_action()

    def get_runtime(self) -> int:
        time = 0
        for char in self.traceback():
            if char == '!':
                time += 3
            else:
                time += 1
```

```
        return time

    def __str__(self):
        return self.traceback()


building = Building(input("Enter file: "))
results = []

def eliminate_actions(actions):
    for action in actions:
        if action.target.occupied is None:
            action.target.occupied = action.origin
            action.conquer(building)
        if action.target.is_end():
            results.append(f"({action.get_runtime():3.0f}s) {action.traceback()}")
            return


Action(building.start).conquer(building)
for j in range(500):
    eliminate_actions(building.schedule[0])
    building.schedule.pop(0)
    building.schedule.append([])
results.sort()
print("\nResult =", results[0])
```