Selles ülesannes oli realiseeritud BFS algoritm ja andmestruktuurid. Kokku programm koonseb 4-st failidest.

Selle ülesanne jaoks võiks olla kasutada juba valmistatud andmestruktuurid nagu list ja dictionary, aga ma otsustastin kasutada oma loodud andmestruktuurid: LavaMap (lava_map.py), mis estab sellest graafi ehk kaarti ja Cell (cell.py), mis esitab sellest graafi tippu ehk kaardi ruutu.

```
@staticmethod
def explore_map(start, lava_map):
    frontier = Queue()
    frontier.put(start)
    came_from = [start]

    while not frontier.empty():
        current = frontier.get()
        for next in lava_map.neighbors(current):
            if next not in came_from and next.get_value() != '*':
                next.set_came_from(current)
                came_from.append(next)
                frontier.put(next)

                if next.get_value() == 'D':
                    lava_map.set_diamond(next)
```

Joonis 1 BFS algoritm

Tee leidmine algust lõpuni toimub 2 etappides. Alguses on vaja avada kogu kaarti (graafi) ja välja selgitada, millised ruudud (tippud) naabrid teineteisele. See loogika on realiseeritud explore_map() meetodis: leiame naabreid, määrame neid frontier-ina ja leiame frontier-i naabreid kuni meie järjekorras on frontier-id. Kui kõik kaart on avatud siis algab 2 etapp, milles me leiame teed lõpust alguseni ja teeme reverse(). See realiseeritud getPath() meetodis.

LavaMap andmestruktuuris oli kirjutatud neighbors() meetod, mis leiab ruudu naabreid. Cell andmestruktuuris on olemas väli self.came_from, mis on link ruudule, millest me käisime.

Koodi võib käivitada main.py failis.