



Vaje iz programiranja: neinformirano preiskovanje

Osnove objektnega programiranja v Pythonu

Pri implementaciji domen bomo potrebovali razrede. Za začetek si pogledjmo primer implementacije razreda v Pythonu:

```
class Pravokotnik:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def ploscina(self):
        return self.a * self.b
```

S tem smo definirali razred. Kot pri drugih jezikih, naredimo instanco razreda in ze lahko kličemo ustrezne metode:

```
p = Pravokotnik(4,5)
print (p.ploscina())
```

Na osnovi primera pa si pogledjmo nekaj posebnosti pri definiranju razredov:

- Razred se definira z besedo `class`.
- Metoda `__init__` (dva podčrtaja na vsaki strani) je konstruktor.
- S `self` dostopamo do razrednih spremenljivk (atributov), podobno kot uporabljamo `this` v javi. Vendar, v Pythonu moramo `self` uporabljati **vedno**, kadar dostopamo do atributov razreda! Prav tako moramo uporabiti `self` vedno kot prvi parameter pri vseh metodah.
- V Pythonu so vse metode tipa `public`.
- Vsakemu razredu lahko definiramo vrsto "posebnih" metod. Npr, če definiramo `__str__(self)`, se bo ta metoda klicala, ko bomo klicali ali `str(...)` oz. `print`. Seznam vseh "posebnih" metod lahko najdete na: [link](#) (glej special methods).

Reševanje ponesrečenja z robotom

S to vajo se boste naučili opisati domeno in razumeli ločevanje opisa domene od implementacije algoritma.

Napisali bomo program, ki bo robota pripeljal od začetnega stanja do ponesrečenja. Najprej bomo potrebovali predstavitev za poligon po katerem se bo robot sprehajal. Uporabili bomo dvodimenzionalno tabelo (1-ke so ovire, 0-ke pa prosta polja):

```
grid = [ [0,0,0,0,0],
          [1,0,1,1,1],
          [0,0,0,0,0],
          [0,0,0,0,0],
          [0,0,0,0,0] ]
```

Napišimo razred za predstavitev te domene. Začnimo s konstruktorjem:

```
class RobotRescue:
    def __init__(self, grid, robot, wounded):
        # 0 - empty space
        # 1 - obstacle
        self.grid = grid
        self.robot = robot
        self.wounded = wounded
```

Tak razred bi inicializirali z: `rr = RobotRescue(grid, [3,3], [0,3])`, kjer je robot na polju (3,3), ponesrečenec pa na (0,3).

Metoda: `solved`

Napiši metodo `solved(self)`, ki vrne `True` v končnem stanju in `False` v nekočnem.

```
rr = RobotRescue(grid, [3, 3], [0, 3])
print(rr.solved())

rr = RobotRescue(grid, [3, 3], [3, 3])
print(rr.solved())
```

Rezultat

```
False
True
```

Rešitev

Metoda: `generate_moves`

Napiši metodo `generate_moves(self)`, ki vrne vse možne "poteze" (vsa možna naslednja stanja). Poteza naj bo par; prvi element predstavlja premik, drugi pa ceno premika (cene naj bodo zaenkrat vedno 1, ker jih še ne bomo potrebovali): `([dy,dx],cena)`. Npr. `([1,0],1)` označuje premik za eno mesto navzdol (št. vrstice se poveča, stolpec ostane enak). V našem začetnem primeru bi pri klicu metode `print (rr.generate_moves())` dobili:

```
[([-1, -1], 1), ([-1, 0], 1), ([-1, 1], 1), ([0, -1], 1), ([0, 1], 1),
 ([1, -1], 1), ([1, 0], 1), ([1, 1], 1)]
```

Rešitev

Metodi `move` in `undo_move`

Napišite metodi `move(self, move_cost)` in `undo_move(self, move_cost)`. Metodi sprejmeta potezo in stanje ustrezno spremenita: `move` nas spravi v naslednje stanje, `undo_move` pa iz novega stanja nazaj v prejšnje.

V PyCharmu je možno testirati program (oz. del programa) tudi v konzoli. Ko napišete metodi `move(self, move_cost)` in `undo_move(self, move_cost)`, označite celotno implementacijo razreda in pritisnite "alt+shift+e". Označena koda se vam je prenesla v konzolo in zdaj lahko testiramo.

```
>>> grid = [[0, 0, 0, 0, 0],
             [1, 0, 1, 1, 1],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0]]
>>> rr = RobotRescue(grid, [3, 3], [0, 3])
>>> moves = rr.generate_moves()
>>> moves
[([-1, -1], 1), ([-1, 0], 1), ([-1, 1], 1), ([0, -1], 1), ([0, 1], 1),
 ([1, -1], 1), ([1, 0], 1), ([1, 1], 1)]
>>> rr.robot
[3, 3]
>>> rr.move(moves[0])
>>> rr.robot
[2, 2]
>>> moves2 = rr.generate_moves()
>>> rr.move(moves2[1])
>>> rr.robot
[2, 1]
>>> rr.undo_move(moves2[1])
>>> rr.undo_move(moves[0])
>>> rr.robot
[3, 3]
```

Rešitev

Preiskovanje

Za našo domeno imamo vse kar potrebujemo za preiskovanje po prostoru stanj.

Preiskovanje v globino

Napišite funkcijo `DF(pos, depth)`, ki za stanje `pos` izvaja iskanje v globino pri maksimalni globini `depth`. Funkcija naj vrne pot od začetnega do končnega stanja. Uporabite rekurzivno varianto, ker jo je lažje sprogramirati. Klic funkcije `DF(rr, 10)` vrne nekaj takega (rešitev s cikli):

```
[[(-1, -1), 1), (-1, -1), 1), (-1, -1), 1), ([0, 1], 1), ([0, -1], 1),
([0, 1], 1), ([0, -1], 1), ([0, 1], 1), ([0, 1], 1), ([0, 1], 1)]
```

Rešitev

Iterativno poglobljanje

Napišite še funkcijo za iterativno poglobljanje `ID(pos)`. Uporabite funkcijo `DF`. Ta funkcija bi na zgornjem primeru morala najti optimalno rešitev:

```
[[(-1, -1), 1), (-1, -1), 1), (-1, 1), 1), ([0, 1], 1)]
```

Uporabite še kakšen drug poligon, npr.:

```
grid = [ [0,1,0,0,0,1],
          [1,0,1,1,1,1],
          [0,0,1,0,0,0],
          [0,1,1,0,0,0],
          [1,0,0,0,0,0],
          [0,0,0,0,0,0] ]
```

Rešitev

NAVIGATION



Home

Site pages

Current course

OUI


■ Participants

Govorilne ure

12 October - 18 October

19 October - 25 October

 Naloga iz neinformiranega preiskovanja

 Rešitev zgornje naloge.

 **Vaje iz programiranja: neinformirano preiskovanje**

26 October - 1 November

2 November - 8 November

9 November - 15 November

16 November - 22 November

23 November - 29 November

30 November - 6 December

7 December - 13 December

14 December - 20 December

4 January - 10 January

11 January - 17 January

18 January - 24 January

Courses
