Battleship

Naslednji program katerega bomo napisali je igra **Battleship** (Potapljanje ladjic).

Ustvarimo **battleship.py** datoteko, v katero bomo pisali naš program.

Za začetek bomo ustvarili funkcijo `create_grid()` in `display_grid( grid )`.

`create_grid()` ustvari igralno ploščo (10 x 10) katero bomo kasneje uporabili. Igralna plošča bo preprost 2D list. Zaenkrat bomo na igralni plošči polja označevali na sledeč način:

- `.` predstavlja prazno mesto (morje)
- `S` predstavlja 1 kos ladje
- `H` predstavlja zadetek
- `M` predstavlja polje kamor smo streljali in nič zadeli

`display_grid( grid )` naj preprosto izpiše našo igralno ploščo na nek lažje berljiv način.

```
R/C 0  1  2  3  4  5  6  7  8  9
0   .  .  .  .  .  .  .  .  .  .
1   .  .  .  .  .  .  .  .  .  .
2   .  .  .  .  .  .  .  .  .  .
3   .  .  .  .  .  .  .  .  .  .
4   .  .  .  .  .  .  .  .  .  .
5   .  .  .  .  .  .  .  .  .  .
6   .  .  .  .  .  .  .  .  .  .
7   .  .  .  .  .  .  .  .  .  .
8   .  .  .  .  .  .  .  .  .  .
9   .  .  .  .  .  .  .  .  .  .
```

Naloga:
Napišite funkciji `create_grid()` in `display_grid( grid )`.

Vizualizacija kode

In [1]:
```
def create_grid():
    # Creates empty grid
    grid = []

    for row in range(10):
        empty_row = []
        for col in range(10):
            empty_row.append(".")
        grid.append(empty_row)
    return grid
```

```
def display_grid(grid):
    print("R/C 0  1  2  3  4  5  6  7  8  9")
    for i, row in enumerate(grid):
        print(f"{i}   ", end="")
        for col in row:
            print(f"{col}", end="  ")
        print()


grid = create_grid()
display_grid(grid)
```

```
R/C 0  1  2  3  4  5  6  7  8  9
0   .  .  .  .  .  .  .  .  .  .
1   .  .  .  .  .  .  .  .  .  .
2   .  .  .  .  .  .  .  .  .  .
3   .  .  .  .  .  .  .  .  .  .
4   .  .  .  .  .  .  .  .  .  .
5   .  .  .  .  .  .  .  .  .  .
6   .  .  .  .  .  .  .  .  .  .
7   .  .  .  .  .  .  .  .  .  .
8   .  .  .  .  .  .  .  .  .  .
9   .  .  .  .  .  .  .  .  .  .
```

Dodajmo funkcijo s katero lahko igralec postavi ladjico na *grid*. Za začetek bomo postavili 1 ladjico - **Patrol Boat**, ki vzeme 2 polji.

Igralec se lahko odloči ali jo bo postavil **horizontalno** ali **vertikalno**.

Naloga:
Napišite funkciji `place_ship()`, ki postavi ladjico **Patrol Boat** dolžine 2 na igralno ploščo.

Vizualizacija kode

In [ ]:
```
def create_grid():
    # Creates empty grid
    grid = []

    for row in range(10):
        empty_row = []
        for col in range(10):
            empty_row.append(".")
        grid.append(empty_row)
    return grid

def display_grid(grid):
    print("R/C 0  1  2  3  4  5  6  7  8  9")
    for i, row in enumerate(grid):
        print(f"{i}   ", end="")
        for col in row:
            print(f"{col}", end="  ")
        print()

# ==========vvvvvv NEW CODE vvvvvv==========
def place_ship(grid, ship_length):
    print(f"Placing ship with length {ship_length}")
    row = int(input("Row: "))
    col = int(input("Col: "))
    orientation = input("[H]orizontal / [V]ertical: ")
    for i in range(ship_length):
        if orientation == "H":
            grid[row][col+i] = "S"
        elif orientation == "V":
            grid[row+i][col] = "S"
    return grid
# ==========^^^^^^ NEW CODE ^^^^^^==========

grid = create_grid()
display_grid(grid)
# ==========vvvvvv NEW CODE vvvvvv==========
grid = place_ship(grid, ship_length=2)
print("After we made a move")
display_grid(grid)
# ==========^^^^^^ NEW CODE ^^^^^^==========
```

Sedaj lahko dodamo vse ladjice za enega igralca:

- **Carrier**, dolžina 5
- **Battleship**, dolžina 4
- **Destroyer**, dolžina 3
- **Submarine**, dolžina 3
- **Patrol Baot**, dolžina 2

Naloga:
Dopolnite program tako, da bo igralec postavil vse svoje ladjice.

```
Carrier, dolžina 5
Battleship, dolžina 4
Destroyer, dolžina 3
Submarine, dolžina 3
Patrol Baot, dolžina 2
```

Vizualizacija kode

In [ ]:
```
def create_grid():
    # Creates empty grid
    grid = []

    for row in range(10):
        empty_row = []
        for col in range(10):
            empty_row.append(".")
        grid.append(empty_row)
    return grid

def display_grid(grid):
    print("R/C 0  1  2  3  4  5  6  7  8  9")
    for i, row in enumerate(grid):
        print(f"{i}   ", end="")
        for col in row:
            print(f"{col}", end="  ")
        print()

def place_ship(grid, ship_length):
    print(f"Placing ship with length {ship_length}")
    row = int(input("Row: "))
    col = int(input("Col: "))
    orientation = input("[H]orizontal / [V]ertical: ")
    for i in range(ship_length):
        if orientation == "H":
            grid[row][col+i] = "S"
        elif orientation == "V":
            grid[row+i][col] = "S"
    return grid


grid = create_grid()
display_grid(grid)
# ==========vvvvvv NEW CODE vvvvvv==========
# Patrol ship
grid = place_ship(grid, ship_length=2)
print("After we made a move")
display_grid(grid)

# Submarine
grid = place_ship(grid, ship_length=3)
print("After we made a move")
display_grid(grid)

# Destroyer
grid = place_ship(grid, ship_length=3)
print("After we made a move")
display_grid(grid)

# Battleship
grid = place_ship(grid, ship_length=4)
print("After we made a move")
display_grid(grid)

# Carrier
grid = place_ship(grid, ship_length=5)
print("After we made a move")
display_grid(grid)
# ==========^^^^^^ NEW CODE ^^^^^^==========
```

Problemi bi se pojavili, kadar dodamo še ladje nasprotnika, saj ne vemo čigava je katera ladja. Problem je, če sta dve ladji preblizu skupaj in ne vemo katera ladja je na katerem mestu. Problem bi se pojavil, če bi naknadno želeli premakniti ladjo, itd..

Zato bi radi vsako ladjico shraniki v nek svoj objekt. Vsako ladjo lahko shranimo v svoj dictionary znotraj katerega bomo hranili:

- **name** - ime ladje

- **length** - dolžino ladje
- **row**, **col** - začetna pozicija ladje
- **orientation** - orientiraznost ladje

Vizualizacija kode

In [ ]:
```
def create_grid():
    # Creates empty grid
    grid = []

    for row in range(10):
        empty_row = []
        for col in range(10):
            empty_row.append(".")
        grid.append(empty_row)
    return grid


# ==========vvvvvv NEW CODE vvvvvv==========
def display_grid(grid, ships):
    for ship in ships:
        for i in range(ship["length"]):
            if ship["orientation"] == "H":
                grid[ship["row"]][ship["col"]+i] = "S"
            elif ship["orientation"] == "V":
                grid[ship["row"]+i][ship["col"]] = "S"
# ==========^^^^^^ NEW CODE ^^^^^^==========
    print("R/C 0  1  2  3  4  5  6  7  8  9")
    for i, row in enumerate(grid):
        print(f"{i}   ", end="")
        for col in row:
            print(f"{col}", end="  ")
        print()

# ==========vvvvvv NEW CODE vvvvvv==========
def place_ship(grid, ship):
    print(f"Placing {ship['name']}, length {ship['length']}")
    ship["row"] = int(input("Row: "))
    ship["col"] = int(input("Col: "))
    ship["orientation"] = input("[H]orizontal / [V]ertical: ")
    return ship
# ==========^^^^^^ NEW CODE ^^^^^^==========


grid = create_grid()
# ==========vvvvvv NEW CODE vvvvvv==========
player1_ships = [{"name": "Carrier", "length": 5, "row":None, "col": None, "orie
                 {"name": "Battleship", "length": 4, "row":None, "col": None, "c
                 {"name": "Destroyer", "length": 3, "row":None, "col": None, "or
                 {"name": "Submarine", "length": 3, "row":None, "col": None, "or
                 {"name": "Patrol Boat", "length": 2, "row":None, "col": None, '
                ]
for i, ship in enumerate(player1_ships):
    player1_ships[i] = place_ship(grid, ship)
    display_grid(grid, player1_ships)
    print()
# ==========^^^^^^ NEW CODE ^^^^^^==========
```

Zadeva je zaenkrat še vredu ampak bi hitro prišlo do zmede, če bi imeli 100, 1000 ladjic.

Lažje bi nam bilo, če bi imeli samo 1 objekt (1 spremenljivko) znotraj katerega bi shranili vse informacije o ladjici (*name, length, row, col, orientation, ...*) in še funkcije, ki vplivajo na te podatke (*place_ship, ...*).

Objektno programiranje

Objektno programiranje je način kako mi združimo medseboj povezane podatke in funkcije, ki delujejo na te podatke.

Primer:

Želimo napisat program "pes".

- Za psa imamo nekja podatkov. Imamo njegovo ime in starost.
- Za psa imamo tudi funkcijo, ki nam ga opiše: f"{ime} je star {starost}"
- Vse to bi radi združili v en objekt

CLASS

Z zadevami, ki so nam znane do sedaj, bi to lahko poizkusili zapakirati v dictionary.

In [ ]:
```
pes = {
    "ime": "Fido",
    "starost": 9,
    "opis": "" # radi bi, da nam kle pove opis psa
}
```

Vse to lahko združimo s pomočjo Class.

todo - tale del je nepotrebn ker sam breaka flow.. sam takoj pokazes kako se definira klas... class imeraztreda, itd..

In [ ]:
```
# Primer: Kako bi izgledal tak class.
# Kaj kera vrstica pomen si bomo pogledal v nadaljevanju
class Pes: #defines the class
    def __init__(self, ime, starost): #special function, dunder functions
        self.ime = ime #creating new variable called name inside our blank objec
        self.starost = starost

    def opis(self):
        return (f'{self.ime} je star {self.starost}')
```

Defining a Class

Začnemo z keyword `class` in nato ime razreda (uporablja se CamelCase način poimenovanja).

```
class ImeRazreda:
    pass
```

Ta razred sedaj predstavlja objekt "Pes". S pomočjo tega razreda lahko sedaj ustvarjamo specifične pse (kjer ima vsak svoje specifično ime, starost, itd.)

Da sedaj ustvarimo našega psa, lahko pokličemo razred in ga shranimo v spremenljivko.

```
x = ImeRazreda()
```

x je sedaj **instanca razreda**.

In [ ]:
```
class Pes:
    pass
```

In [ ]:
```
a = Pes()
print(a)
print(type(a))
```

Znotraj našega Class-a definiramo metodo `__init__()` .

Ko ustvarimo novo instanco našega Class-a (novega specifičnega psa), se pokliče ta metoda.

> `__init__()` se uporablja podobno kot "konstruktor" (iz drugih jezikov), čeprov ni točno konstruktor.

Vizualizacija kode

In [ ]:
```
class Pes:
    def __init__(self):
        print("Ustvarili smo novega psa")

fido = Pes()
print(fido)
print(type(fido))

print()

rex = Pes()
print(rex)
print(type(rex))
```

Znotraj našega razreda lahko ustvarimo spremenljivke, specifične posamezni instanci razreda.

Da ustvarimo spremenljivko specifično instanci razreda se uporabi sledeča sintaksa:

```
self.ime = "Fido"
```

Do spremenljivke dostopamo na sledeč način:

```
print(moj_pes.ime)
Output: Fido
```

V našem primeru bomo vsakemu specifičnemu psu pripisali njegovo ime. Psu bomo ime določili takoj, ko ga ustvarimo. Zato bomo njegovo ime posredovali __init__ metodi.

In [ ]:
```
class Pes: #defines the class
    def __init__(self, ime, starost): #ime, starost so argumenti, k jih posreduj
        self.ime = ime #creating new variable called name inside our blank objec
        self.starost = starost
```

In [ ]:
```
fido = Pes("Fido", 9)
print(fido.ime)
print(fido.starost)
print()

rex = Pes("Rex", 12)
print(rex.ime)
print(rex.starost)
```

`self` parameter je instanca razreda in je avtomatično posredovana kot prvi parameter vsaki metodi našega Class-a.

`self.ime = ime` v naši kodi tako pomeni, da naši instanci pripišemo to spremenljivko. Če bi za razliko napisali `Pes.ime = ime` pa bi spremenljivko name spremenili za celoten Class (ker pa vsi psi nimajo enakega imena ostanemo pri `self.ime = ime`).

> method definitions have self as the first parameter, and we use this variable inside the method bodies – but we don’t appear to pass this parameter in. This is because whenever we call a method on an object, the object itself is automatically passed in as the first parameter. This gives us a way to access the object’s properties from inside the object’s methods.

> In some languages this parameter is implicit – that is, it is not visible in the function signature – and we access it with a special keyword. In Python it is explicitly exposed. It doesn’t have to be called self, but this is a very strongly followed convention.

In [ ]:
```
class Pes: #defines the class
    def __init__(self, ime, starost): #ime, starost so argumenti, k jih posreduj
        print(self) # dobiš isto, k če daš print(fido)
        print(type(self))
        print(id(self))
        self.ime = ime #creating new variable called name inside our blank objec
        self.starost = starost

fido = Pes("Fido", 9)
print(fido.ime)
print(fido.starost)
print()

print(fido)
print(type(fido))
print(id(fido))
```

Naloga:

Ustvarite razred Vozilo. Vsaka instanca naj ima svojo specifično hitrost in kilometrino.

Izpišite njegove lastnosti na sledeč način:

```
"Max hitrost vozila: -hitrost-. Prevozenih je -kilometrina- km."
```

Primeri:

```
Input:
avto = Vozilo(300, 80)

Output:
Max hitrost vozila: 300. Prevozenih je 80 km.

Input:
motor = Vozilo(180, 33)

Output:
Max hitrost vozila: 180. Prevozenih je 33 km.
```

In [ ]:
```
class Vozilo:
    def __init__(self, hitrost, kilometrina):
        self.hitrost = hitrost
        self.kilometrina = kilometrina

avto = Vozilo(300, 80)
motor = Vozilo(180, 33)

print(f"Max hitrost vozila: {avto.hitrost}. Prevozenih je {avto.kilometrina} km.
print(f"Max hitrost vozila: {motor.hitrost}. Prevozenih je {motor.kilometrina} k
```

Znotraj razreda lahko definiramo tudi naše metode s katerimi lahko dostopamo in obdelujemo podatke naših instanc.

Vsaka funkcija/metoda ima najmanj 1 parameter in to je `self`, ki predstavlja instanco razreda in je avtomatično posredovana kot prvi parameter vsaki metodi našega Class-a.

In [ ]:
```
class Pes:
    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

    def opis(self):
        print("Metoda razreda Pes")
```

Vizualizacija kode

In [ ]:
```
class Pes:
    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

    def opis(self):
        print(f"{self.ime} je star {self.starost}")

fido = Pes("Fido", 9)
rex = Pes("Rex", 12)

fido.opis()
rex.opis()
```

Vizualizacija kode

In [ ]:
```
# Se prav mi lahko uporabmo našo instanco objekta in kličemo njeno metodo na sle
fido.opis()

print()

# Oziroma, lahko kličemo direktno Class metodo opis() in sami posredujemo "self"
Pes.opis(fido)
```

Metoda lahko tudi vrne vrednost.

Vizualizacija kode

In [ ]:
```
class Pes:
    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

    def opis(self):
        print(f"{self.ime} je star {self.starost}")

    def vrni_starost(self):
        return self.starost

fido = Pes("Fido", 9)
rex = Pes("Rex", 10)

print(fido.vrni_starost())
print(rex.vrni_starost())

if fido.vrni_starost() > rex.vrni_starost():
    print("Fido je starejši")
else:
    print("Rex je starejši")
```

Naloga:

Ustvarite razred Vozilo. Vsaka instanca naj ima svojo specifično hitrost in kilometrino in **koliko goriva je bilo porabljenega do sedaj**.

Razred Vozilo naj ima funkcija **poraba()**, ki vrne koliko je povprečna poraba tega vozila.

Primeri:

```
Input:
avto = Vozilo(300, 80, 100)
print(f"Vozilo porabi {avto.poraba()}l/km")

Output:
Vozilo porabi 1.25 l/km

Input:
kamion = Vozilo(90, 5500, 125000)
print(f"Vozilo porabi {kamion.poraba()}l/km")

Output:
Vozilo porabi 22.73 l/km
```

Vizualizacija kode

In [ ]:
```python
class Vozilo:
    def __init__(self, hitrost, kilometrina, gorivo):
        self.hitrost = hitrost
        self.kilometrina = kilometrina
        self.gorivo = gorivo

    def poraba(self):
        return self.gorivo / self.kilometrina


avto = Vozilo(300, 80, 100)
print(f"Vozilo porabi {avto.poraba():.2f} l/km")

kamion = Vozilo(90, 5500, 125000)
print(f"Vozilo porabi {kamion.poraba():.2f} l/km")
```

Razredi imajo lahko tudi skupne spremenljivke - spremenljivke, ki so enake vsaki instanci.

> Vsak pes ima 4 noge. Vsak pes ima rad svinjino.

Če želimo, da je spremenljivka enotna celotnemu razredu:

In [ ]:
```python
class Pes:
    hrana = ["svinjina"]
    #set_ = {1,2,3,3,4,5} #sets are modifyable (mutable)

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost
        #self.vrsta += "X"
    def opis(self):
        print(f'{self.ime} je star {self.starost}')
```

Do spremenljivke lahko sedaj dostopamo preko razreda samege:

In [ ]:
```
print(f"Psi najraje jejo {Pes.hrana}")
```

Oziroma, spremenljivka je dostopna preko vsake instance.

In [ ]:
```
fido = Pes("Fido", 9)
rex = Pes("Rex", 10)

print(f'{fido.ime} najraje je {fido.hrana}.')
print(f'{rex.ime} najraje je {rex.hrana}.')
```

Spremenljivko lahko tudi spremenimo in jo tako spremenimo tudi za vse instance razreda.

In [ ]:
```
class Pes:
    hrana = ["svinjina"]
    #set_ = {1,2,3,3,4,5} #sets are modifyable (mutable)

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost
        #self.vrsta += "X"

    def opis(self):
        print(f'{self.ime} je star {self.starost}')

fido = Pes("Fido", 9)
rex = Pes("Rex", 10)

print(f'{fido.ime} najraje je {fido.hrana}.')
print(f'{rex.ime} najraje je {rex.hrana}.')

Pes.hrana = ["teletina"]

print(f'{fido.ime} najraje je {fido.hrana}.')
print(f'{rex.ime} najraje je {rex.hrana}.')
```

Vizualizacija kode

In [ ]:
```
class Pes:
    vrsta = "pes"
    hrana = ["svinjina"]
    #set_ = {1,2,3,3,4,5} #sets are modifyable (mutable)

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost
        #self.vrsta += "X"

    def opis(self):
        print(f'{self.ime} je star {self.starost}')
    def spremeni_vrsto(self, vrsta):
        self.vrsta = vrsta # to nrdi instance variable, ki overwrida Class varia

    def dodaj_hrano(self, hrana):
        self.hrana.append(hrana) # to modify-a variable. In ker je list mutable

    #def add_to_set(self, el):
        #self.set_ = el
        #self.set_.add(el)


fido = Pes("Fido", 9)
rex = Pes("Rex", 10)
ace = Pes("Ace", 3)

print(f'{fido.ime} je {fido.starost} let star in je {fido.vrsta}. Najraje je {fi
print(f'{rex.ime} je {rex.starost} let star in je {rex.vrsta}. Najraje je {rex.h
print(f'{ace.ime} je {ace.starost} let star in je {ace.vrsta}. Najraje je {ace.h

print(30*"*")
Pes.vrsta = "kuščar" # tuki spremenimo variable celotnemu razredu. Vsi, ki so in
fido.spremeni_vrsto("opica") # to naredu self.vrsta = opica za fido instanco raz

rex.dodaj_hrano("teletina")



print(f'{fido.ime} je {fido.starost} let star in je {fido.vrsta}. Najraje je {fi
print(f'{rex.ime} je {rex.starost} let star in je {rex.vrsta}. Najraje je {rex.h
print(f'{ace.ime} je {ace.starost} let star in je {ace.vrsta}. Najraje je {ace.h

#print(30*"*")
#print(Pes.vrsta)
#print(Pes.hrana)
#ace.add_to_set(66)
#print(f'{fido.set_} \n{rex.set_} \n{ace.set_}')
```

Treba bit pozoren, če za spremenljivko instance uporabimo enako ime kot za spremenljivko razreda, potem bo spremenljivka instance override class spremenljivko.

Če je spremenljivka mutable (list, itd..) in jo **modify-amo** (dodajamo elemente, odvzemamo, itd..) potem jo spremenimo za celoten razred.

> When we set an attribute on an instance which has the same name as a class attribute, we are overriding the class attribute with an instance attribute, which will take precedence over it. We should, however, be careful when a class attribute is of a mutable type – because if we modify it in-place, we will affect all objects of that class at the same time. Remember that all instances share the same class attributes:

Battleship

Naloga:

Dopolnimo naš program tako, da mu dodamo class `Ship`. Instanca razreda naj ima sledeče atribute:

- **name** - ime ladje
- **length** - dolžina ladje
- **row**, **col** - začetne koordinate ladje
- **orientation** - ali je ladja obrnjena horizontalno ali vertikalno
- **place_ship** - metoda, s katero nastavimo **row, col, orientation** ladje

Vizualizacija kode

In [ ]:
```python
def create_grid():
    # Creates empty grid
    grid = []

    for row in range(10):
        empty_row = []
        for col in range(10):
            empty_row.append(".")
        grid.append(empty_row)
    return grid



# ==========vvvvvv NEW CODE vvvvvv==========
def display_grid(grid, ships):
    for ship in ships:
        for i in range(ship.length):
            if ship.orientation == "H":
                grid[ship.row][ship.col+i] = "S"
            elif ship.orientation == "V":
                grid[ship.row+i][ship.col] = "S"
    print("R/C 0  1  2  3  4  5  6  7  8  9")
    for i, row in enumerate(grid):
        print(f"{i}   ", end="")
        for col in row:
            print(f"{col}", end="  ")
        print()

class Ship:
    def __init__(self, name, length):
        self.name = name
        self.length = length
        self.row = None
        self.col = None
        self.orientation = None

    def place_ship(self):
        print(f"Placing ship {self.name} with length {self.length}")
        self.row = int(input("Row: "))
        self.col = int(input("Col: "))
        self.orientation = input("[H]orizontal / [V]ertical: ")
# ==========^^^^^^ NEW CODE ^^^^^^==========


grid = create_grid()
# ==========vvvvvv NEW CODE vvvvvv==========
player1_ships = []
player1_ships.append(Ship("Carrier", 5))
player1_ships.append(Ship("Battleship", 4))
player1_ships.append(Ship("Destroyer", 3))
player1_ships.append(Ship("Submarine", 3))
player1_ships.append(Ship("Patrol Boat", 2))

display_grid(grid, player1_ships)

for ship in player1_ships:
    ship.place_ship()
    display_grid(grid, player1_ships)
# ==========^^^^^^ NEW CODE ^^^^^^==========
```

Battleship

grid bomo uporabljali, da hranimo poteze kere smo naredili kot igralec

Naloga:

Dopolnimo naš program tako, da mu dodamo class `Player`.

Razred naj v sebi hrani:

- **name** - ime igralca
- **grid** - igralna plošča, na kateri bomo spremljali poteze katere je igralec naredil
- **ships** - list vseh igralčevih ladji
- **create_grid** - metoda, ki ustvari 2D list, ki predstavlja prazno igralno ploščo
- **display_grid** - metoda, ki izpiše igralčevo igralno ploščo. Na tej plošči bomo spremljali njegove poteze
- **place_ship** - metoda, ki prejme eno Ladjo in jo "postavi" na igralno ploščo
- **display_my_ships** - metoda, ki pokaže igralno ploščo in kje ima igralec postavljene ladje

Vizualizacija kode

In [ ]:
```
class Ship:
    def __init__(self, name, length):
        self.name = name
        self.length = length
        self.row = None
        self.col = None
        self.orientation = None

    def place_ship(self):
        print(f"Placing ship {self.name} with length {self.length}")
        self.row = int(input("Row: "))
        self.col = int(input("Col: "))
        self.orientation = input("[H]orizontal / [V]ertical: ")

# ==========vvvvvv NEW CODE vvvvvv==========
class Player:
    def __init__(self, name):
        self.name = name
        self.grid = self.create_grid()
        self.ships = []
        #self.ships.append(Ship("Carrier", 5))
        #self.ships.append(Ship("Battleship", 4))
        #self.ships.append(Ship("Destroyer", 3))
        #self.ships.append(Ship("Submarine", 3))
        self.ships.append(Ship("Patrol Boat", 2))

        for ship in self.ships:
            self.place_ship(ship)

    def create_grid(self):
        # Creates empty grid
        grid = []
        for row in range(10):
            empty_row = []
            for col in range(10):
                empty_row.append(".")
            grid.append(empty_row)
        return grid

    def display_grid(self):
        print(f"Displaying grid for player {self.name}")
        print("R/C 0  1  2  3  4  5  6  7  8  9")
        for i, row in enumerate(self.grid):
            print(f"{i}   ", end="")
            for col in row:
                print(f"{col}", end="  ")
            print()

    def place_ship(self, ship):
        print(f"Placing a ship for player {self.name}")
        self.display_my_ships()
        ship.place_ship()

    def display_my_ships(self):
        # This is used for debugging
        print(f"Displaying ships for player {self.name}")
        grid = self.create_grid()

        for ship in self.ships:
            for i in range(ship.length):
                if ship.orientation == "H":
                    grid[ship.row][ship.col+i] = "S"
                elif ship.orientation == "V":
                    grid[ship.row+i][ship.col] = "S"
        print("R/C 0  1  2  3  4  5  6  7  8  9")
        for i, row in enumerate(grid):
            print(f"{i}   ", end="")
            for col in row:
                print(f"{col}", end="  ")
            print()
# ==========^^^^^^ NEW CODE ^^^^^^==========



# ==========vvvvvv NEW CODE vvvvvv==========
player1 = Player("Gregor")
print("=========")
player1.display_my_ships()
print("=========")
player1.display_grid()

player2 = Player("Anže")
print("=========")
player2.display_my_ships()
print("=========")
player2.display_grid()
# ==========^^^^^^ NEW CODE ^^^^^^==========
```

Battleship

Naloga:

Dodamo `make_move` metodo našemu Player classu.

Metoda naj od uporabnika pridobi, kam želi ciljati. Nato naj preveri ali je bila ladja zadeta ali ne. V kolikor potrebujete, dopolnite še `Ship` razred.

Vizualizacija kode

In [ ]:

```
class Ship:
    def __init__(self, name, length):
        self.name = name
        self.length = length
        self.row = None
        self.col = None
        self.orientation = None
        # ==========vvvvvv NEW CODE vvvvvv==========
        self.damage = 0
        # ==========^^^^^^ NEW CODE ^^^^^^==========

    def place_ship(self):
        print(f"Placing ship {self.name} with length {self.length}")
        self.row = int(input("Row: "))
```

```python
        self.col = int(input("Col: "))
        self.orientation = input("[H]orizontal / [V]ertical: ")

    # ==========vvvvvv NEW CODE vvvvvv==========
    def check_if_hit(self, row, col):
        ship_coordinates = []
        for i in range(self.length):
            if self.orientation == "H":
                ship_coordinates.append((self.row, self.col+i))
            elif self.orientation == "V":
                ship_coordinates.append((self.row+i, self.col))
        if (row, col) in ship_coordinates:
            self.damage += 1
            return True
        else:
            return False
    # ==========^^^^^^ NEW CODE ^^^^^^==========


class Player:
    def __init__(self, name):
        self.name = name
        self.grid = self.create_grid()
        self.ships = []
        #self.ships.append(Ship("Carrier", 5))
        #self.ships.append(Ship("Battleship", 4))
        #self.ships.append(Ship("Destroyer", 3))
        #self.ships.append(Ship("Submarine", 3))
        self.ships.append(Ship("Patrol Boat", 2))

        for ship in self.ships:
            self.place_ship(ship)

    def create_grid(self):
        # Creates empty grid
        grid = []
        for row in range(10):
            empty_row = []
            for col in range(10):
                empty_row.append(".")
            grid.append(empty_row)
        return grid

    def display_grid(self):
        print(f"Displaying grid for player {self.name}")
        print("R/C 0  1  2  3  4  5  6  7  8  9")
        for i, row in enumerate(self.grid):
            print(f"{i}   ", end="")
            for col in row:
                print(f"{col}", end="  ")
            print()

    def place_ship(self, ship):
        print(f"Placing a ship for player {self.name}")
        self.display_my_ships()
        ship.place_ship()

    def display_my_ships(self):
        # This is used for debugging
        print(f"Displaying ships for player {self.name}")
        grid = self.create_grid()

        for ship in self.ships:
            for i in range(ship.length):
                if ship.orientation == "H":
                    grid[ship.row][ship.col+i] = "S"
                elif ship.orientation == "V":
                    grid[ship.row+i][ship.col] = "S"
        print("R/C 0  1  2  3  4  5  6  7  8  9")
        for i, row in enumerate(grid):
            print(f"{i}   ", end="")
            for col in row:
                print(f"{col}", end="  ")
            print()

    # ==========vvvvvv NEW CODE vvvvvv==========
    def make_move(self, player2):
        print(f"{self.name} making a move.")
        row = int(input("Row: "))
        col = int(input("Col: "))
        for ship in player2.ships:
            if ship.check_if_hit(row, col):
                print("HIT!")
                self.grid[row][col] = "H"
            else:
                print("Miss.")
                self.grid[row][col] = "M"
    # ==========^^^^^^ NEW CODE ^^^^^^==========



player1 = Player("Gregor")
print("=========")
player1.display_my_ships()
print("=========")
player1.display_grid()

player2 = Player("Anže")
print("=========")
player2.display_my_ships()
print("=========")
player2.display_grid()

# ==========vvvvvv NEW CODE vvvvvv==========
for _ in range(3):
    player1.make_move(player2)
    player1.display_grid()
# ==========^^^^^^ NEW CODE ^^^^^^==========
```

In [ ]: