Battleship

Naloga:

Vsakič, ko igralec naredi potezo preverite ali ali imamo zmagovalca ali ne.

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
        self.damage = 0
        # ==========vvvvvv NEW CODE vvvvvv==========
        self.destroyed = False
        # ==========^^^^^^ NEW CODE ^^^^^^==========

    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: ")

    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
            # ==========vvvvvv NEW CODE vvvvvv==========
            self.check_if_destroyed()
            # ==========^^^^^^ NEW CODE ^^^^^^==========
            return True
        else:
            return False

    # ==========vvvvvv NEW CODE vvvvvv==========
    def check_if_destroyed(self):
        if self.damage >= self.length:
            self.destroyed = True
    # ==========^^^^^^ 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()

    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"

    # ==========vvvvvv NEW CODE vvvvvv==========
    def check_if_lost(self):
        for ship in self.ships:
            if not ship.destroyed:
                return False
        return True
    # ==========^^^^^^ 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==========
print(10*"*")
for _ in range(3):
    player1.make_move(player2)
    player1.display_grid()
    if player2.check_if_lost():
        print(f"Player {player1.name} WON!!!")
        break


    player2.make_move(player1)
    player2.display_grid()
    if player1.check_if_lost():
        print(f"Player {player2.name} WON!!!")
        break
# ==========^^^^^^ NEW CODE ^^^^^^==========
```

Sedaj bi radi nadgradili našo igro tako, da malo spremenimo naše ladje.

Spremenili bomo **Submarine** ladjo tako, da se premakne eno pozicijo v levo (oziroma navzdol), vsakič, ko igralec naredi potezo.

Battleship

Naloga:

Dodajte **Submarine** 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
        self.damage = 0
        self.destroyed = False

    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: ")

    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
            self.check_if_destroyed()
            return True
        else:
            return False

    def check_if_destroyed(self):
        if self.damage >= self.length:
            self.destroyed = True


# ==========vvvvvv NEW CODE vvvvvv==========
class Submarine:
    def __init__(self, name, length):
        self.name = name
        self.length = length
        self.row = None
        self.col = None
        self.orientation = None
        self.damage = 0
        self.destroyed = False

    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: ")
```

```
    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
            self.check_if_destroyed()
            return True
        else:
            return False

    def check_if_destroyed(self):
        if self.damage >= self.length:
            self.destroyed = True

    def move(self):
        if self.orientation == "H":
            self.col = 0 if (self.col+self.length+1)> 10 else self.col+1
        elif self.orientation == "V":
            self.row = 0 if (self.row+self.length+1) > 10 else self.row+1
# ==========^^^^^^ 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))
        # ==========vvvvvv NEW CODE vvvvvv==========
        self.ships.append(Submarine("Submarine", 3))
        # ==========^^^^^^ NEW CODE ^^^^^^==========
        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()

    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(f"{ship.name} HIT!")
                self.grid[row][col] = "H"
            else:
                print(f"{ship.name} Miss.")
                self.grid[row][col] = "M"

        # ==========vvvvvv NEW CODE vvvvvv==========
        for ship in self.ships:
            if ship.name == "Submarine":
                ship.move()
        # ==========^^^^^^ NEW CODE ^^^^^^==========

    def check_if_lost(self):
        for ship in self.ships:
            if not ship.destroyed:
                return False
        return True



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==========
print(10*"*")
for _ in range(3):
    player1.make_move(player2)
    player1.display_my_ships()
# ==========^^^^^^ NEW CODE ^^^^^^==========
```

Koda deluje, vendar pa vidimo, da smo jo veliko ponavljali. Namesto tega bi raj nekako že uporabili kodo, katero smo napisali v `Ship` razredu, ter ji dodali še našo funkcjo, ki premakne submarine.

Python Object Inheritance

S pomočjo dedovanja (inheritance) lahko iz že obstoječih razredov ustvarimo nove, bolj specifične razrede.

Tako novo ustvarjeni razredi so imenovani "child classes" in so izpeljani iz "parent classes".

Child-classes podedujejo vse attribute in metode parent-class-a, katere lahko tudi prepišemo (override) ali pa dodamo nove, bolj specifične attribute in metode.

In [ ]:
```
class Pes:
    vrsta = "pes"
    hrana = ["svinjina"]

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

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

    def spremeni_vrsto(self, vrsta):
        self.vrsta = vrsta

    def dodaj_hrano(self, hrana):
        self.hrana.append(hrana)


fido = Pes("Fido", 9)
print(fido.opis())
```

In [ ]:
```
# Sedaj ustvarimo child class, ki bo dedoval iz class Pes

class Bulldog(Pes):
    pass

spencer = Bulldog("Spencer", 15) # ustvarimo novo instanco class Bulldog, ki dea
print(type(spencer)) # vidimo, da je instanca class Bulldog
print(spencer)
```

```
print(spencer.opis()) # vidimo, da smo dedovali metodo opis() iz class Pes
# če deluje metoda opis pol mamo tud .ime in .starost spremenljivko
```

Extending child class

Child class lahko tudi naprej razvijemo z novimi metodami.

Vizualizacija kode

In [ ]:
```
class Bulldog(Pes):
    def bark(self): # dodali smo metodo, ki jo ima samo Bulldog class, ne pa Pes
        return(f'Woof, woof.')
```

In [ ]:
```
spencer = Bulldog("Spencer", 15)
print(spencer.bark())

fido = Pes("Fido", 9)
print(fido.bark())
```

Overriding methods and attributes

Metode in attribute parentclass-a lahko tudi prepišemo.

Vizualizacija kode

In [ ]:
```
class Pes:
    vrsta = "pes"
    hrana = ["svinjina"]

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

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

    def spremeni_vrsto(self, vrsta):
        self.vrsta = vrsta

    def dodaj_hrano(self, hrana):
        self.hrana.append("teletina")


fido = Pes("Fido", 9)
print(fido.opis())
```

In [ ]:
```
class Bulldog(Pes):
    vrsta = "Bulldog"

    def opis(self):
        return f"{self.ime} je star {self.starost} in je {self.vrsta}"

    def bark(self): # dodali smo metodo, ki jo ima samo Bulldog class, ne pa Pes
        return(f'Woof, woof.')
```

```
spencer = Bulldog("Spencer", 15)
print(spencer.vrsta)  # prepisali smo vrsto in sedaj so vsi Bulldogi, vrste Bull
print(spencer.bark()) # še vedno imamo to metodo, ki je specifična za Bulldog cl
print(spencer.opis()) # prepisali smo metodo opis. Sedaj je ta drugačna za class

print()

fido = Pes("Fido", 9)
print(fido.vrsta)
print(fido.opis())
```

Uporaba metod parent class-a

Sedaj želimo dodati najljubši hrano vsakega Bulldoga.

In [ ]:
```
class Pes:
    vrsta = "pes"
    hrana = ["svinjina"]

    def __init__(self, ime, starost):
        self.ime = ime
        self.starost = starost

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

    def spremeni_vrsto(self, vrsta):
        self.vrsta = vrsta

    def dodaj_hrano(self, hrana):
        self.hrana.append("teletina")


fido = Pes("Fido", 9)
print(fido.opis())
```

TO lahko dosežemo tako, da prepišemo __init__ metodo Bulldog class-a:

Vizualizacija kode

In [ ]:
```
class Bulldog(Pes):
    vrsta = "Bulldog"

    def __init__(self, ime, starost, najljubsa_hrana):
        self.ime = ime
        self.starost = starost
        self.najljubsa_hrana = najljubsa_hrana

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

    def bark(self): # dodali smo metodo, ki jo ima samo Bulldog class, ne pa Pes
        return(f'Woof, woof.')

spencer = Bulldog("Spencer", 15, "čevapi")
print(spencer.vrsta)  # prepisali smo vrsto in sedaj so vsi Bulldogi, vrste Bull
print(spencer.bark()) # še vedno imamo to metodo, ki je specifična za Bulldog cl
```

```
print(spencer.opis()) # prepisali smo metodo opis. Sedaj je ta drugačna za class

print()

fido = Pes("Fido", 9)
print(fido.vrsta)
print(fido.opis())
```

Vendar tako ponavljamo določeno kodo:

```
self.ime = ime
self.starost = starost
```

Namesto tega lahko uporabimo *super()* funkcijo s katero dostopamo do metod razreda iz katerega smo dedovali.

In [ ]:
```
class Bulldog(Pes):
    vrsta = "Bulldog"

    def __init__(self, ime, starost, najljubsa_hrana):
        super().__init__(ime, starost)
        self.najljubsa_hrana = najljubsa_hrana

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

    def bark(self): # dodali smo metodo, ki jo ima samo Bulldog class, ne pa Pes
        return(f'Woof, woof.')

spencer = Bulldog("Spencer", 15, "čevapi")
print(spencer.vrsta)  # prepisali smo vrsto in sedaj so vsi Bulldogi, vrste Bull
print(spencer.bark()) # še vedno imamo to metodo, ki je specifična za Bulldog cl
print(spencer.opis()) # prepisali smo metodo opis. Sedaj je ta drugačna za class

print()

fido = Pes("Fido", 9)
print(fido.vrsta)
print(fido.opis())
```

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.

Dodajte **class variable** razredu Vozilo. Spremenljivki naj bo ime **st_gum** in njena vrednost naj bo **4**. Dodajte metodo **opis()**, ki naj izpiše opis vozila.

*Ustvarite podrazreda **Avto** in **Motor**. Razreda naj dedujete od razreda Vozila. Motor razred naj prepiše spremenljivko **st_gum** v **2**. Vsak razred naj pravilno shrani ime vozila, ko ustvarimo novo instanco.*

Primeri:

```
Input:

avto = Avto(300, 80, 500)
avto.opis()


Output:
Max hitrost avto: 300. Prevozenih je 80 km. Poraba vozila je 6.25 l/km.
Vozilo ima 4 gum.


Input:
motor = Motor(90, 220, 520)
motor.opis()

Output:
Max hitrost motor: 90. Prevozenih je 220 km. Poraba vozila je 2.36
l/km. Vozilo ima 2 gum.
```

Vizualizacija kode

In [ ]:
```
class Vozilo:
    st_gum = 4
    vozilo = "vozilo"

    def __init__(self, hitrost, kilometrino, gorivo):
        self.hitrost = hitrost
        self.kilometrina = kilometrino
        self.gorivo = gorivo

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

    def opis(self):
        print(
            f"Max hitrost {self.vozilo}: {self.hitrost}. Prevoženih je {self.kil
        )


class Avto(Vozilo):
    vozilo = "avto"


class Motor(Vozilo):
    vozilo = "motor"
    st_gum = 2


avto = Avto(300, 80, 500)
avto.opis()
```

```
motor = Motor(90, 220, 520)
motor.opis()
```

Multiple inheritance

In [ ]:
```
# Multiple inheritance


class SuperA:
    varA = 10

    def funa(self):
        return 11


class SuperB:
    varB = 20

    def funb(self):
        return 21


class Sub(SuperA, SuperB):
    pass


obj = Sub()
print(obj.varA, obj.funa())
print(obj.varB, obj.funb())

# kle ni problem, ker se nobena stvar ne prekriva (ne instance, ne metode)
```

In [ ]:
```
# Left to right
class A:
    def fun(self):
        print('a')

class B:
    def fun(self):
        print('b')


class C(B,A):
    pass
object_ = C()
object_.fun() # prvo dedujemo iz najbl desnega, pol proti levi in prepisujemo st
```

In [ ]:
```
# override the entities of the same names
class Level0:
    Var = 0
    def fun(self):
        return 0

class Level1(Level0):
    Var = 100
```

```
    def fun(self):
        return 101

class Level2(Level1):
    pass

object_ = Level2() # razred Level0 je parent. Level1 deduje iz Level0 in "overri
print(object_.Var, object_.fun())
```

isinstance() function

s pomočjo funkcijo `python isinstance()` lahko preverimo, če je naša instanca res instanca določenega razreda oziroma razreda, ki od njega deduje.

In [ ]:
```
# override the entities of the same names
class Level0:
    Var = 0
    def fun(self):
        return 0

class Level1(Level0):
    Var = 100
    def fun(self):
        return 101

class Level2(Level1):
    pass

l0 = Level0()
l1 = Level1()
l2 = Level2()

print(isinstance(l2, Level2)) #ali je instanca level2 del razreda Level2
print(isinstance(l2, Level1))
print(isinstance(l2, Level0))
print()
```

inspect.getmro(class_name)

S pomočjo te funkcije lahko izpiše strukturo dedovanja.

In [ ]:
```
import inspect
# override the entities of the same names
class Level0:
    Var = 0
    def fun(self):
        return 0

class Level1(Level0):
    Var = 100
    def fun(self):
        return 101

class Level2(Level1):
    pass
```

```
inspect.getmro(Level2)
```

Banka

Napišite program:

```
class Oseba():
    def __init__(self, ime, priimek):
        # za to instanco naj ustvari spremenljivki ime in priimek

    def opis(self):
        # vrne naj string, znotraj katerega imamo ime in priimek


class Stranka(): # class naj deduje od razreda Oseba()
    def nastavi_stanje(self, stanje):
        # metoda naj ustvari spremenljivko samo za to instaco razreda.
Vrednost naj bo "stanje" oziroma default vrednost naj bo 0. Metoda naj
nato vrne vrednost spremenljivke stanje

    def dvig(self, znesek):
        # Od stanja naj se odšteje znesek.
        # V kolikor ni dovolj denarja na računu naj se dvigne z banke
celotno stanje
        # Na koncu naj metoda vrne dvignjen znesek

    def polog(self, znesek):
        # metoda naj doda velikost zneska stanju
        # nato naj metoda vrne novo stanje


INPUT:
objekt = Stranka("Gregor", "Balkovec")
print(objekt.opis())
print(objekt.nastavi_stanje())
print(objekt.polog(5000))
print(objekt.dvig(2000))
print(objekt.dvig(4000))

OUTPUT:
Gregor Balkovec
0.0
5000.0
2000
Dal ti bom samo 3000.0
3000.0
```

Vizualizacija kode

In [ ]:
```
class Oseba():
    def __init__(self, ime, priimek):
        self.ime = ime
```

```
        self.priimek = priimek

    def opis(self):
        return self.ime + " " + self.priimek


class Stranka(Oseba):

    def nastavi_stanje(self, stanje=0.0):
        self.stanje = stanje
        return self.stanje

    def dvig(self, znesek):
        dvig = 0
        if znesek > self.stanje:
            print("Dal ti bom samo", self.stanje)
            dvig = self.stanje
            self.stanje = 0
        else:
            self.stanje -= znesek
            dvig = znesek
        return dvig

    def polog(self, znesek):
        self.stanje += znesek
        return self.stanje


objekt = Stranka("Gregor", "Balkovec")
print(objekt.opis())
print(objekt.nastavi_stanje())
print(objekt.polog(5000))
print(objekt.dvig(2000))
print(objekt.dvig(4000))
```

Polica

Ustvarite razred `Polica`.

- Vsaka instanca razreda naj ima:
- - **knjige** -> list naslovov knjig, ki se nahajajo na polici
- - **max_knjig** -> integer vrednost, ki pove koliko knjig, gre maximalno na polico
- Ko ustvarimo instanco razreda vanj posredujemo številko maximalnih knjig na polici.
- Ko ustvarimo instanco razreda vanj posredujemo lahko tudi list naslovov knjig, ki se že nahajajo na polici. Če takega seznama ne posredujemo naj ima polica prazen seznam.
- Razred naj ima metodo `kaj_je_na_polici`, ki naj vrne list naslovov knjig
- Razred naj ima metodo `dodaj_knjigo`, ki kot argument prejme string naslova knjige. To knjigo naj doda v list naslovov knjig, če s tem ne presežemo maximalno število knjig. Če bi presegli to število knjige ne dodamo.
- Razred naj ima metodo `uredi_knjige`, ki kot argument **ascending** prejme boolean vrednost, ki nam pove ali naj bodo knjige urejene (glede na prvo črko) v A->Z (vrednost True) oziroma Z->A (vrednost False). Če ta vrednost ni bila

posredovana naj bo default vrstni red A->Z. Metoda naj uredi list naslovov knjig in tega nato vrne

```
polica = Polica(7, ["The Witcher", "Dune", "Harry Potter", "Hamlet",
"Krautov Strojniški Priročnik", "SSKJ"])
print(polica.kaj_je_na_polici())
==> ['The Witcher', 'Dune', 'Harry Potter', 'Hamlet', 'Krautov
Strojniški Priročnik', 'SSKJ']

print(polica.uredi_knjige())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik',
'SSKJ', 'The Witcher']

polica.dodaj_knjigo("Romeo in Julija")
print(polica.kaj_je_na_polici())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik',
'SSKJ', 'The Witcher', 'Romeo in Julija']

polica.dodaj_knjigo("Game of Thrones")
print(polica.kaj_je_na_polici())
==> ['Dune', 'Hamlet', 'Harry Potter', 'Krautov Strojniški Priročnik',
'SSKJ', 'The Witcher', 'Romeo in Julija']

print(polica.uredi_knjige(False))
==> ['The Witcher', 'SSKJ', 'Romeo in Julija', 'Krautov Strojniški
Priročnik', 'Harry Potter', 'Hamlet', 'Dune']
```

Vizualizacija kode

In [ ]:

```
class Polica:
    def __init__(self, max_knjig, knjige=list()):
        self.max_knjig = max_knjig
        self.knjige = knjige

    def kaj_je_na_polici(self):
        return self.knjige

    def dodaj_knjigo(self, knjiga):
        if len(self.knjige) < self.max_knjig:
            self.knjige.append(knjiga)

    def uredi_knjige(self, ascending = True):
        if ascending:
            self.knjige.sort()
        else:
            self.knjige.sort(reverse=True)
        return self.kaj_je_na_polici()



polica = Polica(7, ["The Witcher", "Dune", "Harry Potter", "Hamlet", "Tintin", "
print(polica.kaj_je_na_polici())
print(polica.uredi_knjige())

polica.dodaj_knjigo("Romeo in Julija")
print(polica.kaj_je_na_polici())
```

```
polica.dodaj_knjigo("Game of Thrones")
print(polica.kaj_je_na_polici())
print(polica.uredi_knjige(False))
```

Battleship

Naloga:

Spremenimo naš `Submarine` razred tako, da deduje od `Ship` razreda.

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
        self.damage = 0
        # ==========vvvvvv NEW CODE vvvvvv==========
        self.destroyed = False
        # ==========^^^^^^ NEW CODE ^^^^^^==========

    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: ")

    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
            self.check_if_destroyed()
            return True
        else:
            return False

    def check_if_destroyed(self):
        if self.damage >= self.length:
            self.destroyed = True


# ==========vvvvvv NEW CODE vvvvvv==========
```

```
class Submarine(Ship):
    def move(self):
        if self.orientation == "H":
            self.col = 0 if (self.col+self.length+1)> 9 else self.col+1
        elif self.orientation == "V":
            self.row = 0 if (self.row+self.length+1) > 9 else self.row+1
# ==========^^^^^^ 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(Submarine("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:
            print("Displaying ship", ship.name, "coords:", ship.row, ship.col)
            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()

    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(f"{ship.name} HIT!")
                self.grid[row][col] = "H"
            else:
                print(f"{ship.name} Miss.")
                self.grid[row][col] = "M"

        # ==========vvvvvv NEW CODE vvvvvv==========
        for ship in self.ships:
            if isinstance(ship, Submarine):
                ship.move()
        # ==========^^^^^^ NEW CODE ^^^^^^==========

    def check_if_lost(self):
        for ship in self.ships:
            if not ship.destroyed:
                return False
        return True



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==========
print(10*"*")
for _ in range(3):
    player1.make_move(player2)
    player1.display_my_ships()
# ==========^^^^^^ NEW CODE ^^^^^^==========
```