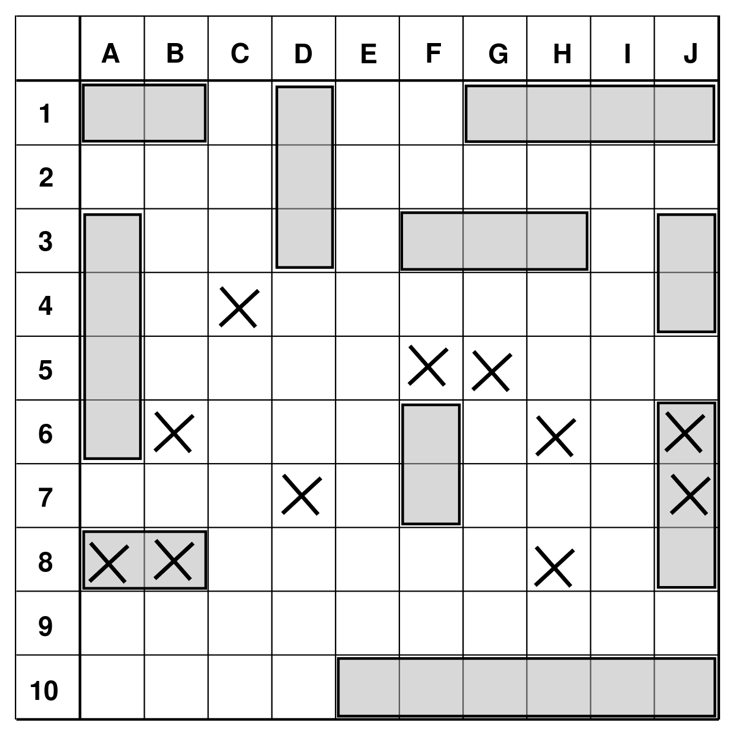


PROJEKTNA NALOGA

POTAPLANJE LADJIC V PYTHONU S TKINTER



Mentor: Klemen Bajec Avtor: Robert Pepelnjak

Ljubljana, šolsko leto 2022/23

# Uvod

Za svojo projektno nalogo sem se odločil izdelati igro potaplanja ladjic s tkinter. Predstavljal sem si, da bo user interface, kjer bosta dve različni tabeli. Ena je tvoja, druga od računalnika. Najprej boš na svojo postavil vse svoje ladje, nato pa boš streljal na računalnikove in hkrati bo tudi računalnik streljal na tvoje.

# Koda

Najprej sem se moral odločiti, kako bom ustvaril ta polja. Precej hitro sem se odločil, da bom naredil grid gumbov, saj se mi je to zdela edina logična rešitev. Vsak od teh gumbov je moral imeti določene lastnosti, katere sem lahko kasneje dostopal. Zato sem najprej definiral class »Knofek.« O classih in na splošno object-oriented programmingu sem se naučil z YouTube posnetkom [na tej povezavi.](https://youtu.be/JeznW_7DlB0)

class Knofek:

def \_\_init\_\_(self, x, y, button,frame):

self.x = x

self.y = y

self.tkButton = button

self.tkFrame = frame

self.stanje = 0

Tako lahko ob nastanku vsakega gumba vidiš njegove koordinate in njegovo stanje. Nato sem zgeneriral tabelo gumbov, ki pripada igralcu:

tabela = []

for y in range(10):

tabela.append([])

Najprej sem ustvaril seznam seznamov, ki bo držal vse koordinate gumbov.

for x in range(10):

frame = tkinter.Frame(okno, width=40, height=40)

button = tkinter.Button(frame, bg="white")

Nato sem definiral nov gumb. Na tak način (ustvarjanje frame-a in nato vezanje gumba nanj) sem se tega lotil, da so vsi gumbi kvadratni, saj lahko frame-u določiš dimenzije s pixli, gumbom pa ne. Nato sem si izmislil objekt tipa knofek, in vezal gumb nanj. Temu gumbu sem nato dodelil funkcijo »klik.«

pametniKnofek = Knofek(x,y,button,frame)

button.configure(command=pametniKnofek.klik)

Ta objekt sem nato ustavil v tabelo.

tabela[y].append(pametniKnofek)

Naslednji del kode sem večinoma ukradel iz stackoverflowa, da so lahko gumbi kvadratni:

frame.grid\_propagate(False)

frame.columnconfigure(0, weight=1)

frame.rowconfigure(0,weight=1)

frame.grid(row=y, column=x)

button.grid(sticky="wens")

S tem sem generiral vse gumbe, ki pripadajo igralcu. Nato sem si moral izmisliti sistem, kako bo postavil ladje. Ladje se lahko med sabo dotikajo, vendar se ne smejo križati. Ena ladja je dolga 5 polij, ena 4, dve 3 in ena 2. Zato sem si izmislil seznam teh ladij:

ladje = [0, 2, 3, 3, 4, 5]

Na začetku je 0 ker sem s tem rešil problem v kodi, do katerega pridemo kasneje.

Ko klikneš na gumb tabele na svoji strani, se bo sprožila funkcija klik. Ta bo gumb obarvala črno in začela iskati, kam bi lahko postavila ladjo.

def klik(self):

self.tkButton.configure(bg="black")

iskanjeladjice(self.x, self.y, tabela)

Funkcija iskanje ladje izgleda takole:

def iskanjeladjice(l\_x, l\_y, kje):

for y in range(10):

for x in range(10):

t = kje[y][x]

t.tkButton.configure(state = tkinter.DISABLED)

if (t.x == l\_x + i and t.y == l\_y) \

or (t.x == l\_x - i and t.y == l\_y) \

or (t.y == l\_y + i and t.x == l\_x) \

or (t.y == l\_y - i and t.x == l\_x):

»kje« sem uporabil zato, da je lahko računalnik uporabil isto funkcijo pri iskanju primernega mesta za ladjo. Zgornji del kode najprej pogleda, če je poljuben gumb pravilne razdalje stran od začetnega gumba. V primeru da je, ga mora še zanimati, če se pot med izvornim gumbom in poljubnim gumbom križa s katerokoli drugo ladjo, ki je tam že postavljena.

Ta pogleda, če ima katerikoli gumb med začetnim gumbom in poljubnim gumbom že nastavljeno stanje. Če ga ima, bo povedal, da se križata.

def a\_se\_kriza(x\_1, x\_2, y\_1, y\_2, kje):

krizanje = False

for vy in range(10):

for vx in range(10):

if (inclusive(vx, x\_1 , x\_2)) and (inclusive(vy, y\_1, y\_2)) and (kje[vy][vx].stanje != 0):

krizanje = True

return krizanje

Če se ne križa potem se bo izvedel naslednji del kode, ki bo vklopil gumb, ki je na pravilnem mestu, mu nastavil barvo na sivo in si shranil njegovo koordinato. Poleg tega mu bo ukaz spremenil na »risiLadjico.«

if not a\_se\_kriza(t.x, l\_x, t.y, l\_y, tabela):

t.drugiKonec = tabela[l\_y][l\_x]

t.tkButton.configure(bg="grey",command=t.risiLadjico, state = tkinter.NORMAL)

Funckija, ki riše ladjico, najprej razglasi par spremenljivk za globalne, da jih lahko spreminjam znotraj funkcije.

def risiLadjico(self):

global napis

global stladje

global i

Vsem gumbom, ki so med določenima dvem koordinatama, nastavi barvo na črno, ter spremeni stanje na »stladje.« Tako shranimo, kateri gumb pripada kateri ladji.

for y in range(10):

for x in range(10):

t = tabela[y][x]

if inclusive(y ,self.drugiKonec.y, self.y) and inclusive(x, self.drugiKonec.x, self.x):

t.tkButton.configure(bg="black", state = tkinter.DISABLED)

t.stanje = stladje

Vse gumbe, ki imajo že nastavljeno stanje pusti na miru:

elif t.stanje != 0:

pass

Ostale pa nazaj prižge. Vse, vključno s tistimi, ki so bili sivi, nastavi nazaj na bele in jim povrne ukaz »klik.«

else:

t.tkButton.configure(

state = tkinter.NORMAL,

bg = "white",

command = t.klik)

Za tem spremenljivki i in stladje spremeni za eno manj:

stladje = stladje - 1

i = ladje[stladje] - 1

Če pa je igralec postavil že vse ladje, se ugasnejo vsi gumbi v tabeli, igralcu je sporočeno, naj strelja, in vsi gumbi, ki pripadajo računalniku, se prižgejo:

if stladje == 0:

ugasn\_vse(tabela)

napis.set("Streljaj!")

for y in range(10):

for x in range(10):

tabela\_racunalnik[y][x].tkButton.configure(state = tkinter.NORMAL)

Naslednji del so računalnikovi gumbi. Najprej jih zgeneriramo, in sicer na identičen način kot igralčeve, le da gumbom nastavimo ukaz »klik\_racunalnik« in po končani generaciji gumbov zaženemo funkcijo imenovano »izmisljotina,« ki si izmisli, kje bo računalnik postavil ladje. Najprej razglasimo par spremenljivk za globalne.

def izmisljotina():

global stladje

global stladje\_racunalnik

global i\_racunalnik

In nato zaženemo kodo, ki se bo ponavljala, dokler računalnik ne postavi vseh ladjic. Najprej si izmisli, ali bo ladjica postavljena vodoravno ali navpično, nato pa si izmisli naključni x in y koordinati.

while stladje\_racunalnik != 0:

katera\_bo = random.randint(0, 1)

naklucn\_x = random.randrange(10)

naklucn\_y = random.randrange(10)

Za tem nastavi drugo koordinto x in y glede na vrednost spremenljivke katera\_bo.

if katera\_bo == 0:

drugi\_x = naklucn\_x + i\_racunalnik

drugi\_y = naklucn\_y

else:

drugi\_y = naklucn\_y + i\_racunalnik

drugi\_x = naklucn\_x

Če sta drugi x ali y izven koordinatnega sistema, se bo koda ponovila še enkrat:

if drugi\_y > 9 or drugi\_x > 9:

continue

Če pa nista, in se pot med koordinatama ne križa s katerokoli že postavljeno ladjo, se izvede naslednji del kode: Računalnik izvede funkcijo, kjer nariše ladjo in spremeni spremenljivke, ki pove katero ladjo bo risal naslednjo in njeno dolžino.

if not a\_se\_kriza(naklucn\_x, drugi\_x, naklucn\_y, drugi\_y, tabela\_racunalnik):

racunalnikRiseLadjico(naklucn\_x, drugi\_x, naklucn\_y, drugi\_y)

stladje\_racunalnik = stladje\_racunalnik - 1

i\_racunalnik = ladje\_racunalnik[stladje\_racunalnik] - 1

Funkcija racunalnikRiseLadjico je precej preprosta: Za vse gumbe, ki se nahajajo med določenima dvema koordinatama, se spremeni stanje na številko ladje, ki jo je postavil. Ne obarva se, saj morajo te ladje biti skrite.

def racunalnikRiseLadjico(x\_1, x\_2, y\_1, y\_2):

global stladje\_racunalnik

for y in range(10):

for x in range(10):

if inclusive(y ,y\_2, y\_1) and inclusive(x, x\_2, x\_1):

tabela\_racunalnik[y][x].stanje = stladje\_racunalnik

Ko klikneš na gumb na računalnikovi strani, se bodo najprej ugasnili vsi gumbi:

def klik\_racunalnik(self):

global ladje\_racunalnik

global zguba

ugasn\_vse(tabela\_racunalnik)

Nato bo pogledal, če je stanje gumba, ki smo ga pritisnili večje od 0, kar pomeni, da se na tem mestu nahaja ladja. V primeru da je tam ladja, bo naredil več stvari. Najprej bo shranil prvotno stanje tega gumba shranil v spremenljivko. Za tem bo temu gumbu spremenil stanje na negativno verzijo sebe. Tako vemo, katere vrsta ladje se nahaja na tem gumbu, poleg tega pa tudi vemo, da je potopljena. Poleg tega bo program v tabeli ladij na mestu, ki pripada tej ladji, zmanjšal število vrednosti za 1. S tem vemo, da tej ladji pripada en manj nepotopljen gumb.

if self.stanje > 0:

grogor = self.stanje

self.tkButton.configure(bg="orange")

ladje\_racunalnik[self.stanje] -= 1

self.stanje \*= (-1)

Tako lahko tudi izvemo, če je ladja bila dokončno potopljena: V primeru da je bila, lahko vse potopljene gumbe, ki pripadajo tej ladji, obarvamo na rdeče. Tako igralec vidi, da je dokončno potopil celo ladjo. Tudi igralcu lahko z napisom sporočimo, katere vrste ladjo je potopil.

if ladje\_racunalnik[self.stanje\*(-1)] == 0:

for y in range(10):

for x in range(10):

if tabela\_racunalnik[y][x].stanje == self.stanje:

tabela\_racunalnik[y][x].tkButton.configure(bg="red")

napis.set("Igralec je potopil %s!" %vrsteLadij[grogor])

Če je seznam ladje\_racunalnik prazen to pomeni, da so vse računalnikove ladje bile potopljene. V tem primeru se igra konča: Ugasnejo se vsi gumbi, računalnikovi in igralčevi. Program tudi sporoči igralcu, da je zmagal.

if sum(ladje\_racunalnik) == 0:

ugasn\_vse(tabela)

ugasn\_vse(tabela\_racunalnik)

napis.set("Zmagal si!")

return

Če pa stanje gumba ni bilo večje od 0 pomeni, da na tem mestu ni ladje in da je igralec zgrešil. Da mu to sporočimo, pobarvamo ta gumb modro:

else:

self.stanje = -99

self.tkButton.configure(bg="blue")

racunalnik\_cilja()

Ko je vse to izvedeno, je na vrsti računalnik. Njegovo obnašanje je zapisano v funkciji racunalnik\_cilja. Ta je sestavljena iz dveh funkcij:

def racunalnik\_cilja():

(y, x) = ideja()

streljaj(y, x)

Najprej se mora računalnik odločiti, kam bo streljal, nato pa mora na to mesto streljati. Kam bo streljal se odloči s funkcijo ideja: Računalnik bo naključno ciljal, dokler ladje ne zadane. Ko jo zadane, bo glede na to koordinato ciljal levo, desno, gor in dol, dokler ne zadane naslednjega dela ladje. Za tem bo v tisto smer nadaljeval, dokler ne zgreši. Če je zgrešil, vendar ladje še ni potopil, potem bo streljal še na drugo stran, dokler ladje ne potopi. Vedno pa se lahko zgodi, da je računalnik v procesu potaplanja ene ladjice zadel še kakšno drugo. Zato moramo pogledati, če je v seznamu zadetih, vendar nepotopljenih ladij kakšna, relativno na katero bi lahko kasneje streljal: V primeru da kaj takega najde, doda v seznam »seznamus« nekaj tuplov, ki jih bom razložil kasneje, ter sporoči, da sedaj cilja na nepotopljeno ladjo.

def ideja():

if potopljena and zadete:

(y\_origin, x\_origin) = zadete[0]

seznamus.extend([(x\_origin + 1, y\_origin, +1, 0), (x\_origin - 1, y\_origin, -1, 0) , (x\_origin, y\_origin + 1, 0, +1), (x\_origin, y\_origin - 1, 0, -1)])

potopljena == False

Naslednji del je ciljanje: Iz seznamusa bo računalnik najprej vzel tuple, ki je na prvem mestu. Če sta obe koordinati med 0 in 9, torej znotraj tabele, in hkrati že nista bili vstreljeni, bo ciljal nanj s kodo return (y, x). Vendar pa potrebuje računalnik tudi pogledati, ali je zadel. V primeru da je zadel, bo naslednjič streljal v isto smer. To doseže s tem, da doda v na začetek seznamusa še en tuple (x+dx, y+dy, dx, dy), kjer sta x in y koordinati, dx in dy pa sta dolžini. Če vzamemo za primer prvi tuple iz zgornjega kosa kode, ki je (x+1, y, +1, 0), je tukaj x = x+1, y=y, dx=1 in dy=0. Če bo z uporabo tega tupla zadel ladjo in dodal tuple (x+dx, y+dy, dx, dy), bo v bistvu le povečal x za 1, y pa pustil na miru. V smer x+1 bo streljal, dokler bo zadeval ladjo, nato pa bo šel na naslednji tuple v seznamu »seznamus:«

while seznamus:

(x, y, dx, dy) = seznamus.pop(0)

if inclusive(x, 0, 9) and inclusive(y, 0, 9) \

and ((x, y) not in ze\_ciljane):

if (tabela[y][x].stanje > 0 or inclusive(tabela[y][x].stanje, -5, -1 )):

if inclusive(tabela[y][x].stanje, -5, -1 ):

seznamus.insert(0, (x+dx\*2,y+dy\*2,dx,dy))

else:

seznamus.insert(0, (x+dx,y+dy,dx,dy

return (y, x)

V primeru, da je že zadel ta gumb, bo skočil čezenj in ciljal naslednjega. Če je seznam »seznamus« prazen to pomeni, da računalnik nima pojma kje bi lahko bila kakšna ladjica. V tem primeru si bo izmislil naključno koordinato in nanj streljal. Če bo zadel, bo spet dopolnil seznam »seznamus« s štirimi tupli, enim za vsako smer, kamor lahko strelja relativno na zadeti gumb.

else:

nakljucna\_koordinata()

if tabela[y\_origin][x\_origin].stanje > 0:

seznamus.extend([(x\_origin + 1, y\_origin, +1, 0), (x\_origin - 1, y\_origin, -1, 0) , (x\_origin, y\_origin + 1, 0, +1), (x\_origin, y\_origin - 1, 0, -1)])

potopljena = False

return (y\_origin, x\_origin)

To vse se bo ponavljalo, dokler računalnik ali igralec ne potopita vseh ladij svojega nasprotnika. Takrat se bodo vsi gumbi izključili in igra se bo zaključila.

# Zaključek

Z izdelavo tega projekta sem se naučil veliko, predvsem uporabo classov in to, kako pomembne so funkcije. Veliko mi je tudi pomagalo, da sem si osnovne ideje skiciral na list papirja, preden sem jih šel programirati. Tako nisem programiral popolnoma na slepo, vendar sem imel vsaj osnovno idejo, kako se bom problema lotil. S končnim rezultatom projekta sem zelo zadovoljen in vesel sem, da sem si zadal nekaj tako zahtevnega.