**9. praktiskais darbs. 2. semestris**

**1. uzdevums**

Sastādīt programmu, kas nodrošina kvadrātiskas matricas element ievadi, aprēķina visu “diagonāļu”, kas paralēlas galvenai diagonālei summu nodrukā uz ekrāna pašu matricu un “tās diagonāļu” elementu summu.

Izvades formāts:

1 2 3 |

4 5 6 | 3

7 8 9 | 8

---------------

7 12 | 15

**Kods:**

# Programmas nosaukums: Galveno diagonaļu summa matricai

# 1. uzdevums (1MPR09\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas nodrošina kvadrātiskas matricas elementu ievadi, aprēķina visu "diagonāļu", kas paralēlas

# galvenai diagonālei summu nodrukā uz ekrāna pašu matricu un "tās diagonāļu" elementu summu.

# Izvades formāts:

# 1 2 3 |

# 4 5 6 | 3

# 7 8 9 | 8

# ━━━━━━━━━━━━━━━

# 7 12 | 15

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def ievade(n, m):

# Prasa lietotājam ievadīt visas matricas elementus. Matricas elementi var būt tikai vesēlie skaitļi.

# Ja tiek ievadīt nevesels skaitlis vai simbolu virkne, tad prasa ievādīt tikmēr, kamēr neievadīs pareizi.

# Atgriež NumPy masīvu ar lietotāja ievadītajiem veseliem elementiem.

# n - matricas (divdimensiju masīva) rindas skaits

# m - matricas (divdimensiju masīva) kolonnas skaits

a = numpy.empty((n, m))

for i in range(n):

for j in range(m):

t = input("Ievadiet matricas elementu a(" + str(i) + "," + str(j) + ") ===> ")

while is\_whole(t) == False:

t = input("Kļūda! Ievadītam skaitlim jābūt veselam!\nIevadiet matricas elementu a(" + str(i) + "," + str(j) + ") ===> ")

t = int(t)

a[i, j] = t

# Atgriezt NumPy masīvu ar ievadītiem veseliem elementiem

return a

def izvade\_matricu\_ar\_diagonalu\_summam(a):

# Atgriež simbolu virkni, kura repzenetē "glītu diagonāļu summu". Labi atspoguļo tikai skaitļus līdz 999.

# Piemēram:

# 3 -1 -2 |

# 2 3 4 | -2

# 5 6 7 | 3

# ━━━━━━━━━━━━━━━

# 5 8 | 13

# a - divdimensijas masīvs (kvadrātiska matrica), kurai sameklējam diagonaļu summu izmantojot funkciju "diagonalu\_summa"

# a.ndim # dimensiju skaits

# a.shape # kortežs ar masīva izmēriem

n = a.shape[0] # x axis

m = a.shape[1] # y axis

s = ""

c = diagonalu\_summa(a)

for i in range(n):

for j in range(m):

s = s + "{:4d}".format(int(a[i, j]))

if i == 0:

s = s + " | \n"

len1 = len(s) - 3

continue

s = s + " | " + str(int((c[-1 \* i]))) + "\n"

s = s + " " + "━" \* (len1) + "\n"

k = 0

for i in range(n):

if i == 0:

s = s + " "

continue

s = s + " " + str(int((c[i])))

k += 1

s = s + " | " + str(int((c[k + 1])))

return s

def diagonalu\_summa(a):

# Atgriež divdimensijas masīvu, kur labājā un apaksējā stūri elementi atspoguļo kvadrātiskas matricas diagonaļu summu

# Ja nav iespējams aprēķināt, tad atgriež "Kļūda"

# a - divdimensijas masīvs (kvadrātiska matrica)

n = a.shape[0]

m = a.shape[1]

if m == n:

c = numpy.zeros(2 \* n)

for k in range(1, 2 \* n):

s = 0

if k <= n:

# Apakšsumma

for j in range(k):

s = s + a[n - k + j, j]

else:

# augšsumma

for j in range(k - n, n):

s = s + a[n + j - k, j]

c[k] = s

return c

else:

return "Kļūda"

def is\_natural(n):

# Pārbauda vai simbolu virkne ir naturāls skaitlis vai nav

# Ja ir naturāls skaitlis, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

if str(n).isdigit() and float(n) == int(n) and int(n) > 0:

return True

else:

return False

def is\_whole(n):

# Pārbauda vai simbolu virkne ir vesels skaitlis vai nav

# Ja ir vesels skaitlis, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

try:

n = int(n)

except:

return False

else:

return True

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

n = input("Ievadiet kvadrātiskas matricas izmēru ===> ")

while is\_natural(n) == False:

n = input("Kļūda! Ievadītam skaitlim jābūt naturālam!\nIevadiet kvadrātiskas matricas izmēru ===> ")

n = int(n)

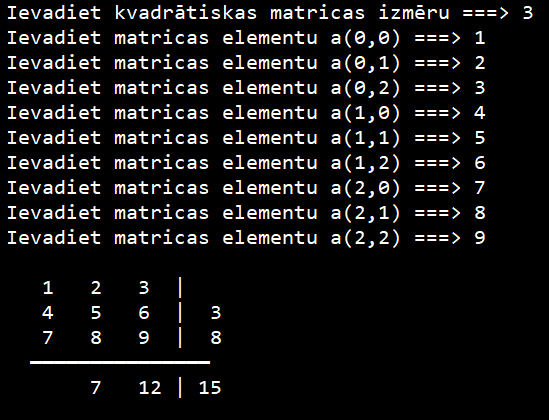
a = ievade(n, n)

print("")

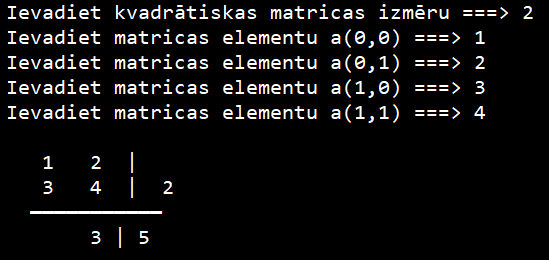
print(izvade\_matricu\_ar\_diagonalu\_summam(a))

**Testa piemēri:**

1)



2)



3)

Text

Description automatically generated

4)

Calendar

Description automatically generated with low confidence

5)

A picture containing graphical user interface

Description automatically generated

6)

Text

Description automatically generated

7)

Graphical user interface, text

Description automatically generated

8)

A picture containing calendar

Description automatically generated

9)

A screen shot of a computer

Description automatically generated with low confidence

10)

Calendar

Description automatically generated with medium confidence

**2. uzdevums**

Sastādīt programmu, kas realizē teātra biļešu iegādi lidzīgi kā “Biļešu paradīze”. Ar šādiem papildus atvieglojumiem:

1. teātra skatītaju zālē ir N rindu un katrā rindā M sēdvietu, M un N vērtības ievāda lietotājs
2. uzsākot iepirkšanos apmēram 50% biļešu ir pārdotas
3. vienā reizē var nopirkt tikai vienu biļeti, ievadot izvēlēto rindas un sēdvietas numuru
4. ja izvēlēta sēdvieta ir brīva, tā tiek atzīmēta kā aizņemta un pāriet pie nākamās biļetes iegādes, bet, ja izvēlēta vieta ir aizņemta, tad uz ekrāna tiek parādīta informācija par visām sēdvietām un pieprasīta atkārtota sēdvietas izvēle
5. Biļešu iegāde tiek atkārtota, kamēr lietotājs ievada 0.rindu un 0.sēdvietu vai visas biļetes ir izpārdotas.

**Kods:**

# Programmas nosaukums: "Bilešu paradīze" (tekstuāls režims)

# 2. uzdevums (1MPR09\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē teātra bilešu iegādi līdzīgi kā "Bilešu paradīze".

# Ar šādiem papildus atvieglojumiem.

# 1. teātra skatītāju zālē ir N rindu un M sēdvietu. M un N vērtības ievada lietotājs.

# 2. uzsākot iepirkšanos apmēram 50% biļešu ir pārdotas.

# 3. vienā reizē var nopirkt tikai vienu biļeti, ievadot izvēlēto rindas un sēdvietas numuru.

# 4. ja izvēlēta sēdvieta ir brīva, tā tiek atzīmēta kā aizņemta un pāriet pie nākamās biļetes iegādes, bet, ja izvēlēta vieta ir aizņemta,

# tad uz ekrāna tiek paradīta informācija par visām sēdvietām un pieprasīta atkārtota sēdvietas izvēle.

# 5. Biļešu iegāde tiek atkārtta, kāmer lietotājs ievada 0.rindu un 0.kolonnu vai visas biļetes ir izpārdotas.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

import random

def create\_random\_2array(n, m):

# Atgriež divdimensiju masīvu, kur visi elementi ir vai nu 0, vai 1 un aptuvēni 50% no visu elementu skaita ir 1.

# n - divdimensiju masīva rindu skaits

# m - divdimensiju masīva kolonnu skaits (sēdvietu skaits)

a = numpy.ones((n, m))

num\_ones = int(n \* m \* 0.5)

ones\_count = 0

for i in range(n):

for j in range(m):

if ones\_count < num\_ones and random.random() < 0.5:

a[i][j] = 0

ones\_count += 1

return a

def add\_numeration\_to\_matrix(a):

# Pievieno rindas un kolonnu numerāciju matricai un atgriež simbolu virkni ar matricas elementiem un ir rindas un kolonnas numerācija

# a - divdimensijas masīvs

s = ""

len1 = a.shape[1]

for j in range(len1 + 1):

s = s + "{:3d}".format(j) # izmantot formāta norādītāju, lai nodrošinātu nepieciešāmu atstarpi

s += "\n" # pievienot jaunu rindiņu pēc pirmās rindas

n = a.shape[0]

for i in range(n):

s = s + "{:3d}".format(i + 1) # rindas numurs

for j in range(len1):

s = s + " " + a[i, j]

#s = s + "{:3.0f}".format(a[i, j])

s += "\n" # pievienot jaunu rindiņu pēc pirmās rindas

return s

def choose\_what\_change\_to\_one(a, sk):

# Biļetes pirkšanai. Ja viss ir izpārdots (ja visi ir 1), tad print("Izpārdots!\n")

# Var izvelēties, kuru ciparu izmainīt uz 1.

# 1 -> 1 (vieta aizņemta)

# 0 -> 1 (print("\nBiļete nopirkta!\n"))

# a - divdimensijas masīvs (matrica)

# sk - vieninieku skaits. Ja ir lielāks nekā n x m, tad izpārdots.

n, m = a.shape

if sk >= n \* m:

print("\nŠodien viss ir izpārdots!\n")

return

while True:

r = input("Ievadiet vēlamo rindas numuru biļetes nopirkšanai ==> ")

while not is\_natural\_or\_zero(r):

r = input("Kļūda! Ievadiet vēlamo naturālo rindas numuru biļetes nopirkšanai ==> ")

r = int(r)

s = input("Ievadiet vēlamo sēdvietu numuru ==> ")

while not is\_natural\_or\_zero(s):

s = input("Kļūda! Ievadiet vēlamo naturālo sēdvietu numuru biļetes nopirkšanai ==> ")

s = int(s)

try:

if r == 0 and s == 0:

return print("Paldies, ka izmantojāt mūsu pakalpojumus!")

elif r == 0 or s == 0:

print("\nNekorekta ievade!\n")

print(add\_numeration\_to\_matrix(replace\_ones\_to\_a\_zeros\_to\_b(a)))

elif a[r - 1, s - 1] == 1:

print("\nJusu izvēlēta vieta " + str(r) + ".rindā " + str(s) + ".sēdvietā ir aizņemta!\n")

print(add\_numeration\_to\_matrix(replace\_ones\_to\_a\_zeros\_to\_b(a)))

elif a[r - 1, s - 1] == 0:

a[r - 1, s - 1] = 1

print("\nBiļete nopirkta! Jūsu vieta ir " + str(r) + ".rindā " + str(s) + ".sēdvietā.\n")

print(add\_numeration\_to\_matrix(replace\_ones\_to\_a\_zeros\_to\_b(a)))

sk += 1

if sk >= n \* m:

print("Šodien viss ir izpārdots!\n")

print(add\_numeration\_to\_matrix(replace\_ones\_to\_a\_zeros\_to\_b(a)))

return

except IndexError:

print("\nIr jāievāda reālu sēdvietu!\nUz redzēšanos!")

quit()

def count\_ones(a):

# Saskaita vieninieku skaitu noteiktā masīvā a un atgriež to int skaitļi

# a - divdimensijas masīvs (matrica)

count = 0

for i in range(a.shape[0]):

for j in range(a.shape[1]):

if a[i, j] == 1:

count += 1

return count

def replace\_ones\_to\_a\_zeros\_to\_b(arr):

# Izmaina visus divdimensija masīva vieniniekus to A, bet nulles to B

# 1 -> A

# 0 -> B

# arr - divdimensijas masīvs (matrica)

new\_arr = numpy.empty(arr.shape, dtype='str')

new\_arr[arr == 1] = 'A'

new\_arr[arr == 0] = 'B'

return new\_arr

def is\_natural(n):

# Pārbauda vai simbolu virkne ir naturāls skaitlis vai nav

# Ja ir naturāls skaitlis, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

if str(n).isdigit() and float(n) == int(n) and int(n) > 0:

return True

else:

return False

def is\_natural\_or\_zero(n):

# Pārbauda vai simbolu virkne ir naturāls skaitlis vai nulle, vai nav

# Ja ir naturāls skaitlis vai nulle, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

if str(n).isdigit() and float(n) == int(n) and int(n) >= 0:

return True

else:

return False

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

n = input("Ievadiet kopējo rindu skaitu teātrī ==> ")

while not is\_natural(n):

n = input("Kļūda! Ievadiet kopējo naturālu rindu skaitu teātrī ==> ")

n = int(n)

m = input("Ievadiet kopējo sēdvietu skaitu teātrī ==> ")

while not is\_natural(m):

m = input("Kļūda! Ievadiet kopējo naturālu rindu skaitu teātrī ==> ")

m = int(m)

a = create\_random\_2array(n, m)

b = add\_numeration\_to\_matrix(replace\_ones\_to\_a\_zeros\_to\_b(a))

print("")

sk = count\_ones(a)

print(b)

print("Sveicinām teātri!\nŠeit var nopirkt biļeti uz izrādi!\n\nIziešanai no sistēmas ievadiet rindas numuru: 0, sēdvietu: 0")

choose\_what\_change\_to\_one(a, sk)

**Testa piemēri:**

1)

A picture containing text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

5)

Text

Description automatically generated

A picture containing background pattern

Description automatically generated6)

7)

Text

Description automatically generated

8)

Text

Description automatically generated

9)

Text

Description automatically generated

10)

Text

Description automatically generated

11)

Text

Description automatically generated

12)

Text

Description automatically generated

13)

Text

Description automatically generated

**3. uzdevums**

Sastādīt programmu, kas realizē labirinta izveidi un apstaigāšanu atbilstoši lekcijā dotajiem nosacījumiem, papildus paredzot, ka labirintā 5% no rūtiņu kopskaita ir necaurejami šķēršļi.

**Kods:**

# Programmas nosaukums: Labirints ar necaurejami šķēršļiem.

# 3. uzdevums (1MPR09\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas realizē labirinta izveidi un apstaigāšanu atbilstoši lekcijā dotajiem nosacījumiem,

# papildus paredzot, ka labirinta 5% no rūtiņu kopskaita ir necaurejami šķēršļi.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import numpy

def is\_labirint\_path(i, j, lab, tr):

# Atgriež masīvu found\_path (atrasts ceļš), kurš sastāv no frāzem "Pa labi", "Uz leju".

# Rekursīvi atrod ceļu, lai izietu no labirinta (var iet tikai no mazākas vērtības uz lielāku (vai vienādu)).

# i - 0 (parasti)

# j - 0 (parasti)

# lab - divdimensijas masīvs ar labirinta segmēnta vērtībam.

# tr - trase

# trase = numpy.empty(path\_length, "O")

# path\_length = n + m - 2

n = lab.shape[0] - 1

m = lab.shape[1] - 1

if (i == n) and (j == m):

return True

else:

found\_path = False

if (i < n) and (lab[i + 1, j] >= lab[i, j]):

tr[i + j] = "Uz leju"

found\_path = is\_labirint\_path(i + 1, j, lab, tr)

if not found\_path and (j < m) and (lab[i, j + 1] >= lab[i, j]):

tr[i + j] = "Pa labi"

found\_path = is\_labirint\_path(i, j + 1, lab, tr)

return found\_path

def izvade(a):

# Atgriež simbolu virkni, kura reprezentē noteikto matricu, kā simbolu virkni stābiņveidā.

# -1 pārkonvertē kā X.

# X - nejauši šķēršļi (-1)

# a - divdimensijas masīvs (matrica)

# a.ndim # dimensiju skaits

# a.shape # kortežs ar masīva izmēriem

n = a.shape[0] # x axis

m = a.shape[1] # y axis

s = "\n"

for i in range(n):

for j in range(m):

if a[i, j] == -1:

s = s + " X"

else:

s = s + "{:4d}".format(int((a[i, j])))

s = s + "\n"

return s

def get\_column(a):

# a - viendimensijas masīvs

# [1, 2, 3] konvertēsies

# 1

# 2

# 3

# Atgriež str masīva elementus, kā kolonnu

col\_str = ""

for elem in a:

col\_str = col\_str + str(elem) + "\n"

return col\_str

def random\_negative\_fill(a, chance):

# chance = 0.05 (5%) no kopēja skaita būs šķēršļi (-1)

# chance - skaitlis (float). Jo lielāka šī vērtība, jo lielāka iespēja, ka būs vairāk "segmēntu" ar -1.

# a - divdimensijas masīvs (labirints)

# Atgriež divdimensijas masīvu (matricu) kurš ir aizpildīts ar -1.

# -1 skaits ir chance \* masīva\_kopējais\_elementu\_skaits (chance == 0.05 (5%))

a\_shape = a.shape

# Aprēķinat ievietojamo negatīvo vērtību skaitu

num\_negatives = int(a\_shape[0] \* a\_shape[1] \* chance)

# Izvēlieties nejaušus rindu un kolonnu "segmēntus", lai ievietotu negatīvas vērtības

indices = numpy.zeros((num\_negatives, 2), dtype=int) # indices ir nulles masīvs, kura izmērs ir (num\_negatives, 2), un tajā tiek glabāti izvēlēto "segmētu" rindu un kolonnu koordinātas.

# Pēc nejauši izvēlēto koordinātu noteikšanas row\_idx un col\_idx, tiek pārbaudīts, vai tās nav jau aizpildītas ar -1, un

# vai tās nav sākumpunkts (0, 0) vai beigu punkts (a\_shape[0] - 1, a\_shape[1] - 1), kas atbilst labirinta sākumam un beigām.

# Ja nosacījumi tiek izpildīti, koordinātas tiek ievietotas masīvā indices un while cikls tiek pārtraukts.

for i in range(num\_negatives):

while True: # Tiek izmantots while cikls, lai izvēlētos "segmētu" koordinātas, kas nav jau iepriekš aizpildītas ar -1.

row\_idx = numpy.random.randint(a\_shape[0])

col\_idx = numpy.random.randint(a\_shape[1])

if a[row\_idx, col\_idx] != -1 and not (row\_idx == 0 and col\_idx == 0) and not (row\_idx == a\_shape[0] - 1 and col\_idx == a\_shape[1] - 1):

indices[i] = (row\_idx, col\_idx)

break

# Aizpildit nejauši izvēlētus "segmēntus" ar -1

row\_indices = indices[:, 0]

col\_indices = indices[:, 1]

for i in range(len(row\_indices)):

row\_idx = row\_indices[i]

col\_idx = col\_indices[i]

a[row\_idx, col\_idx] = -1

return a

def is\_natural(n):

# Pārbauda vai simbolu virkne ir naturāls skaitlis vai nav

# Ja ir naturāls skaitlis, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

if str(n).isdigit() and float(n) == int(n) and int(n) > 0:

return True

else:

return False

def is\_natural\_or\_zero(n):

# Pārbauda vai simbolu virkne ir naturāls skaitlis vai nulle, vai nav

# Ja ir naturāls skaitlis vai nulle, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

if str(n).isdigit() and float(n) == int(n) and int(n) >= 0:

return True

else:

return False

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

n = input("Ievadi labirinta garumu ===> ")

while not is\_natural(n):

n = input("Kļūda! Ievadiet naturālu labirinta garumu ==> ")

n = int(n)

m = input("Ievadi labirinta platumu ===> ")

while not is\_natural(m):

m = input("Kļūda! Ievadiet naturālu labirinta platumu ==> ")

m = int(m)

labirints = numpy.empty((n, m))

path\_length = n + m - 2

trase = numpy.empty(path\_length, "O")

for i in range(n):

for j in range(m):

t = input("Labirints(" + str(i) + "," + str(j) + ") ===> ")

while not is\_natural\_or\_zero(t):

t = input("Kļūda! Ievadiet naturālu (vai 0) labirinta segmēnta vērtību!\nLabirints(" + str(i) + "," + str(j) + ") ===> ")

labirints[i, j] = t

print("\nJusu labirints:")

print(izvade(labirints))

print("Jusu labirints ar nejaušiem šķēršļiem (5% no kopēja skaita):")

a = random\_negative\_fill(labirints, 0.05)

print(izvade(a))

if is\_labirint\_path(0, 0, a, trase):

print("Labirintā var virzoties tikai no mazākas vērtības uz lielāku (vai vienādu)!\nVirzoties var tikai uz leju vai pa labi!")

print("Labirintu var iziet virzoties tā:")

print(get\_column(trase))

else:

print("Labirints nav izejams!")

**Testa piemēri:**

1)

Text

Description automatically generated

2)

Text

Description automatically generated

3)

Text

Description automatically generated

4)

Text

Description automatically generated

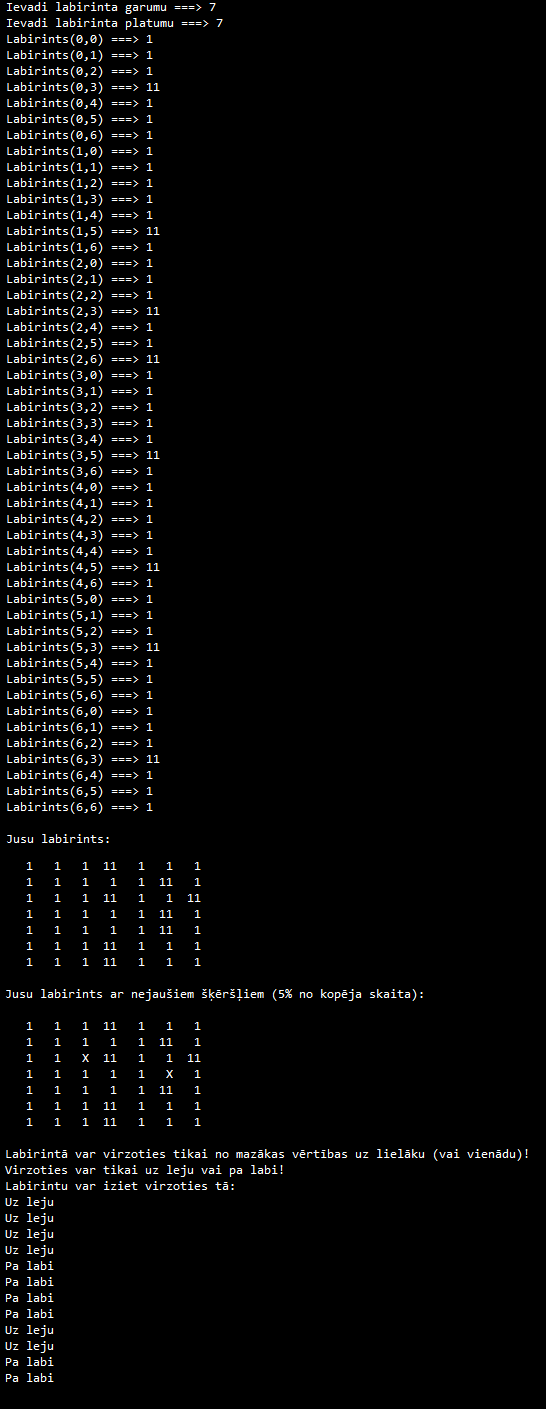
5)

Text

Description automatically generated

Text

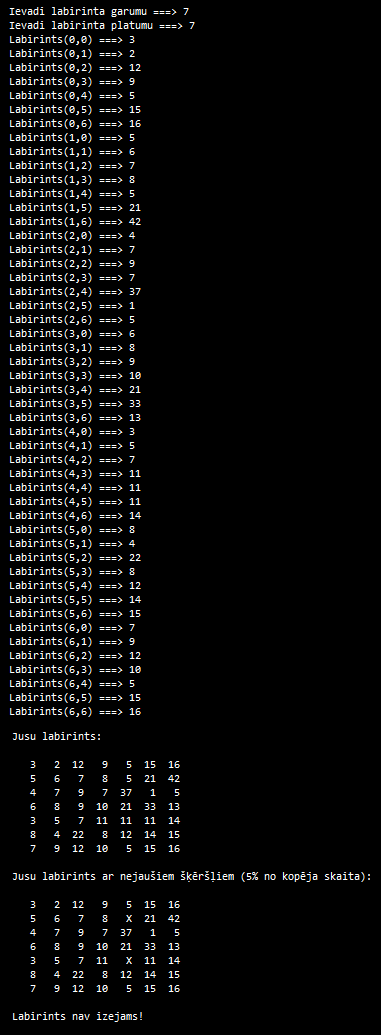
Description automatically generated with low confidence6)

7)

8)

Text

Description automatically generated

9)

**PU1. uzdevums**

Realizēt 2. uzdevumu par biļešu iegādi tā, ka uz ekrāna ir redzamas visas sēdvietas un ar peles klikšķi (klikšķiem) var izvēlēties vienu (vairākas) biļetes.

**Kods:**

# Programmas nosaukums: "Biļešu paradīze" ar GUI

# Papilduzdevums 1 (1MPR09\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Realizēt uzdevumu par biļešu iegādi tā, ka uz ekrāna ir redzamas visas sēdvietas un ar peles klikšķi (klikšķiem) var izvēlēties vienu (vairākas) biļetes.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

import tkinter

import numpy

import random

# global mainīgo saraksts:

# root, a, label2, buy\_button

def result():

# Komanda tiek izsaukta pēc "submit\_button" nospiešanas ( = ).

# Paņemt n un m no lietotāja un izveido matricu, kur aptuvēni 50% no matricas ir 1.

# Izveido globālu mainīgu a un tas ir tās matrica.

# Izsauc komandu show\_array()

global a

n = input\_rindas\_skaits.get()

m = input\_kolonnas\_skaits.get()

n = int(n)

m = int(m)

arr = numpy.ones((n, m))

num\_ones = int(n \* m \* 0.5)

ones\_count = 0

for i in range(n):

for j in range(m):

if ones\_count < num\_ones and random.random() < 0.5:

arr[i][j] = 0

ones\_count = ones\_count + 1

a = arr

show\_array(a)

def toggle\_button(poga):

# Pogas sēdvietas izvelēšanai

# 1 -> 1 "Vieta aizņemta!"

# 0 -> X "Biļete izvelēta!"

# X -> 0 "Biļete atcelta!"

# check\_button\_text() pārbauda vai uz visām pogām ir uzrakstīts "1". (pilnā pārlase).

# Ja uz visiem, tad blokēt pogas un uzrakstīt kā visas biļetes ir izpārdotas.

if poga["text"] == "0":

poga["text"] = "X"

label2.config(text="Biļete izvelēta!")

check\_button\_text()

elif poga["text"] == "1":

poga["text"] = "1"

label2.config(text="Vieta aizņemta!")

check\_button\_text()

elif poga["text"] == "X":

poga["text"] = "0"

label2.config(text="Biļete atcelta!")

check\_button\_text()

def change\_button\_text():

# Pārlasa visas pogas root'ā un izmainā "X" uz "1" (izvelētus uz aizņemts)

# Izmainām label2 tekstu uz "Nopirkts!"

# Tiek izmantots globals mainīgais root

global root

for poga in root.grid\_slaves(): # Parlasam visas pogas

if type(poga) == tkinter.Button:

if poga.cget("text") == "X": # cget dot iespēju pārbaudit vai teksts uz pogas ir "X"

poga.configure(text="1")

label2.config(text="Nopirkts!")

check\_button\_text()

def check\_button\_text():

# Izmantojot divus karogus, pārbaudam vai uz visiem pogam ir uzrakstīts "1". (Pogas pilnā pārlase)

# Ja uz visām ir "1", tad visas pogas bloķēt un uzrakstīt "Visas biļetes ir izpārdotas!"

all\_ones = True

has\_two = False # two == X (jo pirmkārts tika izmantots 0 - brīvs, 1 - aizņemts, 2 - izvelēts)

for poga in root.grid\_slaves(): # Pilnā pārlase pogiem un label'iem kuri ir grid'ā izmantojot in root.grid\_slaves() (atrādu internētā tādu funkciju)

if poga.cget("text") == "1 - aizņemta vieta\n0 - brīva vieta\nX - izvelētas vietas pirkšanai\nIzvēlieties vietas pirkšanai!":

continue

if poga.cget("text") == "Nopirkt izvelētas biļetes":

continue

if poga.cget("text") == "Nopirkts!":

continue

if poga.cget("text") == "Biļete izvelēta!":

continue

if poga.cget("text") == "X":

has\_two = True

if poga.cget("text") != "1":

all\_ones = False

if all\_ones:

label3 = tkinter.Label(root, text="Visas biļetes ir izpārdotas!", font=("Arial", 12), anchor="ne")

label3.place(relx=1.0, rely=0.0, anchor="ne")

buy\_button.config(state="disabled")

for poga in root.grid\_slaves(): # grid\_slave() palīdz visas pogas bloķēt

poga.config(state="disabled")

elif not has\_two:

buy\_button.config(state="disabled")

else:

buy\_button.config(state="normal")

def button\_callback(poga):

# Definē callback() funkciju kas izsauc toggle\_button(poga). Jā tieši tā neuzrakstīt, tad programma nestrādās

def callback():

toggle\_button(poga)

return callback

def show\_array(a):

# Izveido jaunu logu "root", un izveido tajā pogas (sēdvietas) nepieciešamā skaitā, ar komandu button\_callback

# Parada jaunu logu "root".

# Tiek izveidota poga "Nopirkt izvelētas biļetes", pēc tas nospiešanas tiek izsaukta komanda change\_button\_text

# Pogas ir dizaktivētas (disabled) pēc noklusējumā.

# label2 ir izveidots tukšs un globāls, lai pēc tam to mainītu uz "Nopirkts!" vai uz "".

# label4 ir izveidots informatīvam nolūkam

# tiek izveidoti trīs globāli mainīgie

global root, label2, buy\_button

root = tkinter.Tk()

root.title("Teātris")

rindas = len(a)

kolonnas = len(a[0])

pogas = []

for i in range(rindas):

for j in range(kolonnas):

poga = tkinter.Button(root, text=str(int(a[i][j])), width=2, height=1)

poga.config(command=button\_callback(poga))

poga.grid(row=i, column=j)

pogas.append(poga)

label2 = tkinter.Label(root, text="")

label2.grid(row=i + 1, column=j + 1)

label4 = tkinter.Label(root, text="1 - aizņemta vieta\n0 - brīva vieta\nX - izvelētas vietas pirkšanai\nIzvēlieties vietas pirkšanai!")

label4.grid(row=i + 3, column=j + 3, sticky="s")

buy\_button = tkinter.Button(root, text="Nopirkt izvelētas biļetes", width=20, bd=1, command=change\_button\_text, state="disabled")

buy\_button.grid(row=i + 2, column=j + 2)

check\_button\_text()

root.mainloop()

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

logs = tkinter.Tk() # Tkinter (lai izmantotu to komandas)

logs.geometry("170x130") # Loga izmēra definēšana

logs.title("Biļešu nopirkšana") # Windows "loga" nosaukums

# Labels teātrim

label\_teatris = tkinter.Label(logs, text="Teātris", font=("Arial", 12))

label\_teatris.place(x=20, y=6)

label\_teatra\_izmers = tkinter.Label(logs, text="Teātra izmērs:", font=("Arial", 12))

label\_teatra\_izmers.place(x=20, y=30)

label\_x = tkinter.Label(logs, text="x", font=("Arial", 15))

label\_x.place(x=56, y=58)

# Entry logi

input\_rindas\_skaits = tkinter.Entry(logs, width=3)

input\_rindas\_skaits.place(x=30, y=65)

input\_kolonnas\_skaits = tkinter.Entry(logs, width=3)

input\_kolonnas\_skaits.place(x=75, y=65)

# -----------------------------------------

submit\_button = tkinter.Button(logs, text="=", width=20, bd=1, command=result) # Izmantojam definētas komandas, lai pēc pogas nospiešanas tā komanda tiek izpildīta

submit\_button.place(x=110, y=62, width=25) # Parādam, kur poga tiks attēlota

logs.mainloop() # lai logs būtu redzāms visu laiku

**Testa piemēri:**

1. Video testa piemērs

<https://youtu.be/uvh5oh2g2IQ>

2)

Graphical user interface, application

Description automatically generated

3)

Graphical user interface, application

Description automatically generated

4)

Graphical user interface, text, application

Description automatically generated

5)

Graphical user interface, text, application

Description automatically generated

6)

Graphical user interface, text, application

Description automatically generated

7)

Graphical user interface, application

Description automatically generated

8)

Graphical user interface, text, application

Description automatically generated

9)

Graphical user interface, text, application

Description automatically generated

10)

Graphical user interface, text, application

Description automatically generated

11)

Graphical user interface, application

Description automatically generated

12)

Graphical user interface, application

Description automatically generated

13)

A picture containing diagram

Description automatically generated

A picture containing table

Description automatically generated14)

**PU2. uzdevums**

Realizēt kuģīšu šaušanas spēli jeb spēli KARTUPELIS – 2 spēlētāji dators un lietotājs.

1. Līmenis abu spēlētāju kuģu izvietojumu ievada lietotājs un uz ekrāna redzamas tikai lietotāja gājieni datora “jūrā” un lietotāja “jūra ar kuģiem” un datora izdarītiem gājieniem. Katrs secīgi var izdarīt tikai vienu gājienu.
2. Līmenis – dators pats izvieto savus kuģus “jūrā”
3. Līmenis – dators šāvienus izdara plānveidīgi nevis uz labu laimi.

**Kods:**

**Testa piemēri:**

1)