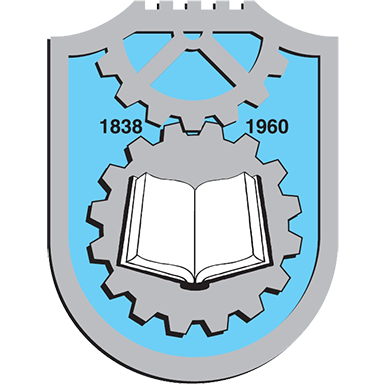
Универзитет у Крагујевцу

Факултет инжењерских наука



**Софтверски инжењеринг**

Семинарски рад:

Препознавање тумора мозга са 2D снимака DICOM формата

Студент: *Алекса Лазаревић* Предметни наставник: *др Филиповић Ненад*

Садржај

[1. Увод 3](#_Toc1)

[2. Задатак 3](#_Toc2)

[3. Софтверски алати 4](#_Toc3)

[4. Структура пројекта и код 4](#_Toc4)

[4.1 Фајл: detekcija.py 4](#_Toc5)

[4.2 Фајл: dicomhandler.py 10](#_Toc6)

[4.3 Фајл: ui.py 11](#_Toc7)

[4.4 Фајл: tumor\_prepoznavanje.py 18](#_Toc8)

[5. UML дијаграми 18](#_Toc9)

[5.1 Дијаграм случајева коришћења (use case) 18](#_Toc10)

[5.2 Дијаграм секвенци 19](#_Toc11)

[5.3 Дијаграм активности 20](#_Toc12)

[5.4 Дијаграм стања 21](#_Toc13)

[5.5 Дијаграм класа 22](#_Toc14)

[6. Литература 23](#_Toc15)

# Увод

Тумори представљају један од најозбиљнијих и најкомпликованијих проблема медицини у 21. веку. Брза прогноза озбиљних тумора је први корак у спречавању даљег раста, и успешног лечења пацијента. Закашњење прогнозе тумора често доводи до великих проблема опасних по живот. Код тумора лобањске дупље, односно тумора мозга, за разлику од осталих типова тумора, и доброћудни (бенигни) и злоћудни (малигни) су практично једнако опасни јер у оба случаја повећавају притисак унутар лобањске шупљине, и доводе до тешког неуролошког испада, и у најгорем случају су смртног исхода. Хитна неурохируршка интервенција је посебно битна за ове типове тумора. Циљ програма би био убрзавање дијагнозе тумора директним читањем са DICOM формата.

# Задатак

Почетак сваког софтверског пројекта састоји се из јасног састављања задатка, ондносно проблема који сам софтвер решава. Конретно, у овом случају, проблем можемо представити у више тачака:

* Учитавање DICOM фајлова, извлачење потребних података, и претварање медицинске слике из фајла у формат за даљу употребу,
* Конструкција корисничког интерфејса лаког за употребу,
* Проналажење контуре тумора на основу претходно дефинисаних параметара (доњи и горњи праг одлуке, минимална и максимална површина тумора),
* Проналазак добрих почетних (default) параметара који раде у већини случајева.

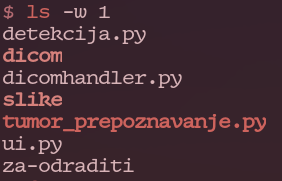
# Софтверски алати

Пре реализације изворног кода, представљамо софтверске алате који ће бити у употреби. Као програмски језик, биће у употреби **Python 3.10.6**, за писање кода **neovim** текст едитор, и за покретање референтну имплементацију **CPython**. Од библиотека, коришћени су:

* **TkInter**, библиотека уграђена у референтној имплементацији Python-a, служи за прављење корисничких интерфејса,
* **opencv-python**, библиотека са алатима корисним за манипулацију графике односно слика,
* **matplotlib**, библиотека за приказивање графика и слика,
* **pydicom**, библиотека за читање DICOM фајл формата.
* **os**, библиотека уграђена у референтној имплементацији Python-а, служи за разне системске функције (*нпр. читање или чување фајлова, фолдера*).

# Структура пројекта и код

Сам код програма расподељен је у више фајлова. На слици 1 можемо видети да се у директоријуму налазе 3 **python** фајлова, и 2 директоријума.



Слика 1 – директоријум пројекта

# Фајл: detekcija.py

У овом фајлу се налази све везано за саму детекцију тумора и приказивање резултата детекције. Сам фајл представља класу *Detekcija*. Ова класа има 3 функције, од којих је једна констрактор. Констрактор ове класе прима путању до слике на којој ће детекција тумора да се изврши. Констрактор чита слику и чува је у *ndarray* формату у варијаблу *origImg*. Код констрактора се види на блоку кода 1.

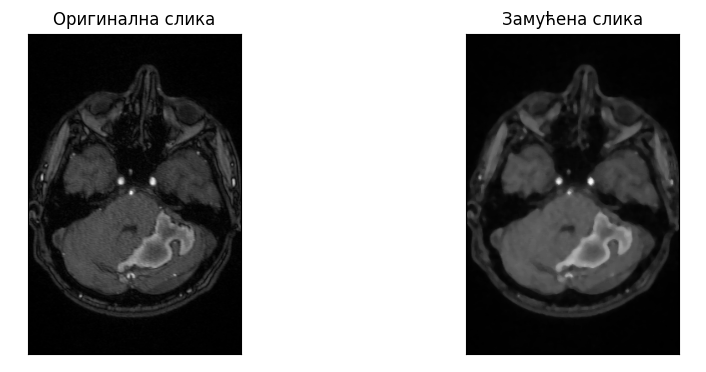
def \_\_init\_\_(self, putanjaDoSlike):

print(f"Изабран фајл: {putanjaDoSlike}, читам...")

self.\_\_origImg = cv2.imread(putanjaDoSlike, 0)

Блок кода 1 – Констрактор класе Detekcija

Поред констрактора, ова класа садржи и функције *detektujTumor* и *prikaziRezultate*. Функција detektujTumor прима доњи и горњи бинарни праг, као и минималну и максималну површину тумора. Функција прво узима слику и извршава *cv.medianBlur* на њу, а резултат чува у варијаблу *blurredImg* у класиПоред констрактора, ова класа садржи и функције detektujTumor и prikaziRezultate. Замућивањем слике истичемо појаву тумора на мозгу, па касније лакше можемо добити бинарним прагом одговарајуће резултате (*слика 2*). Код ове функције налази се на блоку кода 2.



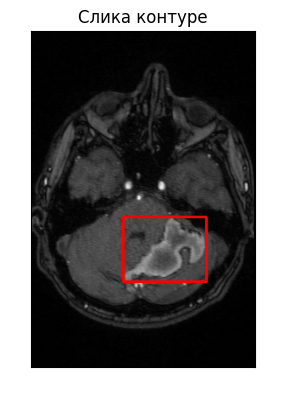
Слика 2 – Замућена слика и оригинална слика

Након замућивања, извршавамо бинарни праг[[1][2]](#_Литература)на замућену слику (са слике 2), и подешавањем параметара доњег и горњег прага, добијамо резултат као на слици 3. Ову слику чувамо у параметру *tresholdedImg* у класи.



Слика 3 – Слика бинарног прага

Након овог препроцесирања слике, остало нам је да извучемо контуре добијене бинарним прагом, да пронађемо највећу контуру[[3]](#_Литература) и да је означимо на слици. Поред тога, морамо да проверимо да површина највеће контуре не прелази одређену границу површине, као и да није мања од одређене границе површине. Резултат ове операције нам даје где се налази тумор на слици. Исцртавамо контуру на још једној копији слике и ту слику чувамо у параметру *output* у класи. Пример резултата ове операције видимо на слици 4.



Слика 4 – Слика контуре тумора

Комплетан код ових операција можемо видети на блоку кода 2.

def detektujTumor(self, donjiThreshold, gornjiThreshold, minPovrsina, maxPovrsina):

self.\_\_blurredImg = self.\_\_origImg.copy()

print(f"Копирам слику и претварам у RGB за контуру касније...")

self.\_\_output = cv2.cvtColor(self.\_\_blurredImg.copy(), cv2.COLOR\_GRAY2RGB)

print(f"Замућујем слику...")

self.\_\_blurredImg = cv2.medianBlur(self.\_\_blurredImg, 5)

print(f"Извршавам бинарни праг на слику...")

ret, self.\_\_thresholdedImg = cv2.threshold(self.\_\_blurredImg, donjiThreshold,

gornjiThreshold, cv2.THRESH\_BINARY)

print(f"Проналазим највећу контуру на слици...")

contours, hierarchy = cv2.findContours(

self.\_\_thresholdedImg, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE)

c = max(contours, key=cv2.contourArea)

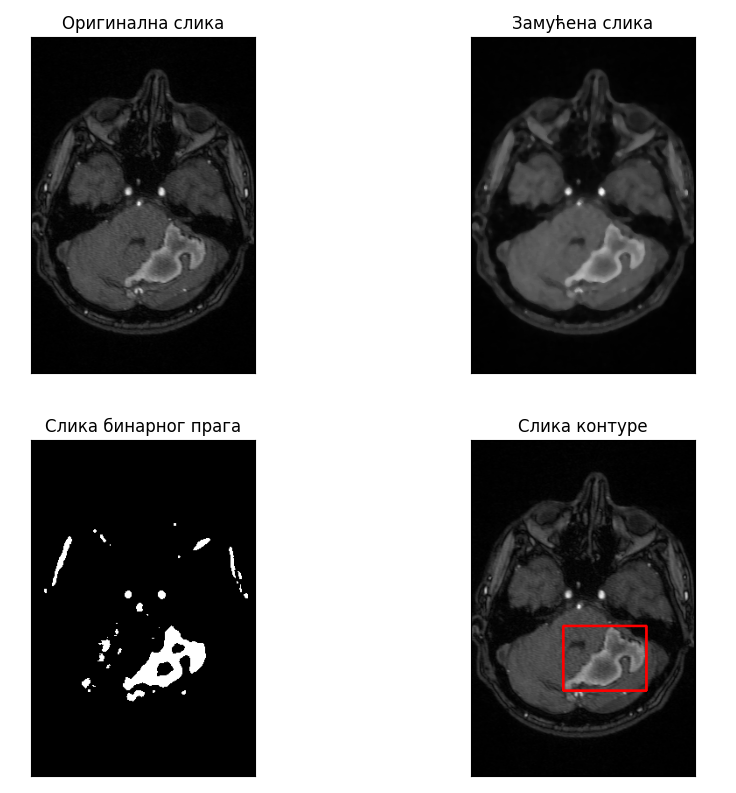
if minPovrsina < cv2.contourArea(c) < maxPovrsina:

x, y, w, h = cv2.boundingRect(c)

cv2.rectangle(self.\_\_output, (x, y), (x + w, y + h), (255, 0, 0), 2)

Блок кода 2 – Функција detektujTumor

На крају у овом фајлу, имамо функцију *prikaziRezultate*, која не прима параметре, али користи сачуване слике и приказује[[6]](#_Литература) их кориснику помоћу matplotlib библиотеке. Резултат операције можемо видети на слици 5. Код ове функције је у блоку кода 3.



Слика 5 – Комплетан приказ операција, приказано функцијом prikaziRezultate

def prikaziRezultate(self):

print(f"Приказујем слике...")

titles = ["Оригинална слика", "Замућена слика",

"Слика бинарног прага", "Слика контуре"]

images = [self.\_\_origImg, self.\_\_blurredImg, self.\_\_thresholdedImg, self.\_\_output]

for i in range(len(images)):

plt.subplot(2, 2, i + 1)

plt.imshow(images[i], 'gray')

plt.title(titles[i])

plt.xticks([])

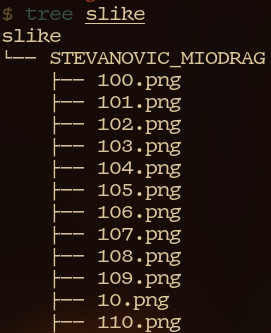
plt.yticks([])

plt.show()

Блок кода 3 – Функција prikaziRezultate

# Фајл: dicomhandler.py

У овом фајлу се налази класа *DicomHandler*, која врши све операције везане за DICOM фајл формат. Конкретно ова функција има само једну функцију. Ова функција прима *listboxPacijent*, у ком треба да се прикаже листа пацијената. Функција пролази кроз све у директоријуму[[7]](#_Литература) *dicom*, креира директоријум *slike*, унутар тог креира директоријум са називом пацијента, и на крају у директоријуму са називом пацијента претвара све одговарајуће DICOM фајлове у слике. Поред тога, очисти *listboxPacijent* од ставки, и исписује нове ставке (имена пацијената) у исти. Резултат креираног фолдера можемо видети на слици 6, а функцију *napraviSlike* у блоку кода 4.



Слика 6 – Директоријум слика креиран од стране програма

**Напомена**: пошто *PatientName* у имену садржи “^” уместо размака, мењамо ово доњим цртама када правимо директоријум.

def napraviSlike(listboxPacijent):

print("Правим директоријум ./slike ако не постоји...")

if not os.path.exists("slike"):

os.makedirs("slike")

print("Проналазим све DICOM фајлове у ./dicom директоријуму...")

for f in os.listdir("./dicom"):

fajl = os.path.join('dicom', f)

print(f'Пронађен фајл: {str(fajl)}')

dataset = pydicom.dcmread(fajl)

putanja = f'slike/{str(dataset.PatientName).replace("^", "\_")}'

if not os.path.exists(putanja):

os.makedirs(putanja)

print(f'Чувам слику: {putanja}/{f}.png...')

plt.imsave(f'{putanja}/{f}.png', dataset.pixel\_array, cmap=plt.cm.gray)

listboxPacijent.delete(0, END)

for pacijent in os.listdir("./slike"):

listboxPacijent.insert(END, pacijent)

Блок кода 4 – функција napraviSlike

# Фајл: ui.py

У овом фајлу се налази све везано за кориснички интерфејс, односно једна класа под називом *UIClass*. Ова класа се састоји из 8 функција, од којих је једна констрактор класе. Констрактор је јако једноставан и врши само 3 функције од којих је једна за прављење корисничког интерфејса, друга за повезивање догађаја (нпр. изабир пацијента у listBox-у) и позива mainloop[[4]](#_Литература). За библиотеку израде корисничког интерфејса изабран је TkInter, који долази уз сваку референтну инсталацију python-a, дакле у већини случајева ово није неопходно инсталирати. Констрактор се налази у блоку кода 5.

def \_\_init\_\_(self):

self.napraviUI()

self.napraviBindove()

self.\_\_window.mainloop()

Блок кода 5 – Констрактор класе UIClass

Следећа функција је *napraviUI*, и служи за креирање целог корисничког интерфејса. Кориснички интерфејс је направљен из 3 колона, од којих прва служи за бирање пацијента и слике, друга за приказ изабране слике, и коначно трећа за постављање параметара препознавања тумора. Сви креирани атрибути се чувају унутар класе, да би се касније употребљивали унутар осталих функција класе. Функција не прима параметре, и не враћа никакву вредност. Програмски код ове функције видимо на блоку кода 6, а пример корисничког интерфејса (боје зависе од теме система) видимо на слици 7.

def napraviUI(self):

self.\_\_window = tk.Tk()

self.\_\_window.geometry('800x600')

self.\_\_window.resizable(False, False)

self.\_\_window.columnconfigure(0, weight=1)

self.\_\_window.columnconfigure(1, weight=2)

self.\_\_window.columnconfigure(2, weight=1)

self.\_\_window.title(

'Препознавање тумора мозга на DICOM формату - Алекса Лазаревић ')

self.\_\_pacijenti = ()

if os.path.exists("./slike"):

self.\_\_pacijenti = os.listdir("./slike")

self.\_\_kolona1 = tk.Frame(self.\_\_window)

self.\_\_kolona1.grid(column=0, row=0)

self.\_\_labelPacijent = tk.Label(self.\_\_kolona1, text="Изабери пацијента: ")

self.\_\_labelPacijent.grid(column=0, row=1)

self.\_\_labelPacijent.configure(anchor='n')

self.\_\_listboxPacijent = tk.Listbox(self.\_\_kolona1, listvariable=tk.StringVar(

value=self.\_\_pacijenti), width=30, height=6, selectmode='single')

self.\_\_listboxPacijent.grid(

column=0, row=2)

self.\_\_labelSlike = tk.Label(self.\_\_kolona1, text="Изабери слику: ")

self.\_\_labelSlike.grid(column=0, row=3)

self.\_\_labelSlike.configure(anchor='n')

self.\_\_listboxSlike = tk.Listbox(

self.\_\_kolona1, width=30, height=6, selectmode='single', exportselection=False)

self.\_\_listboxSlike.grid(

column=0, row=4)

self.\_\_canvasSlike = tk.Canvas(self.\_\_window, width=300, height=400)

self.\_\_canvasSlike.grid(column=1, row=0)

self.\_\_kolona3 = tk.Frame(self.\_\_window)

self.\_\_kolona3.grid(column=3, row=0)

self.\_\_dugmeNapraviSlike = tk.Button(self.\_\_kolona1, text="Направи слике",

command=self.napraviSlikeHandler)

self.\_\_dugmeNapraviSlike.grid(column=0, row=0)

self.\_\_dugmeNapraviSlike.grid(column=0, row=0)

self.\_\_labelDonjiThreshold = tk.Label(self.\_\_kolona3, text="Доњи праг: ")

self.\_\_labelDonjiThreshold.grid(column=0, row=0)

self.\_\_labelDonjiThreshold.configure(anchor='n')

self.\_\_donjiThresholdRange = tk.Scale(self.\_\_kolona3, from\_=0, to=255, orient='horizontal')

self.\_\_donjiThresholdRange.set(80)

self.\_\_donjiThresholdRange.grid(column=0, row=1)

self.\_\_labelGornjiThreshold = tk.Label(self.\_\_kolona3, text="Горњи праг: ")

self.\_\_labelGornjiThreshold.grid(column=0, row=2)

self.\_\_labelGornjiThreshold.configure(anchor='n')

self.\_\_gornjiThresholdRange = tk.Scale(self.\_\_kolona3, from\_=0, to=255, orient='horizontal')

self.\_\_gornjiThresholdRange.set(87)

self.\_\_gornjiThresholdRange.grid(column=0, row=3)

self.\_\_labelminimalnaPovrsina = tk.Label(self.\_\_kolona3, text="Минимална површина тумора: ")

self.\_\_labelminimalnaPovrsina.grid(column=0, row=4)

self.\_\_labelminimalnaPovrsina.configure(anchor='n')

self.\_\_minimalnaPovrsinaTumora = tk.Scale(self.\_\_kolona3, from\_=0, to=200, orient='horizontal')

self.\_\_minimalnaPovrsinaTumora.set(100)

self.\_\_minimalnaPovrsinaTumora.grid(column=0, row=5)

self.\_\_labelmaksimalnaPovrsina = tk.Label(self.\_\_kolona3, text="Максимална површина тумора: ")

self.\_\_labelmaksimalnaPovrsina.grid(column=0, row=6)

self.\_\_labelmaksimalnaPovrsina.configure(anchor='n')

self.\_\_maksimalnaPovrsinaTumora = tk.Scale(self.\_\_kolona3, from\_=200, to=5000, orient='horizontal')

self.\_\_maksimalnaPovrsinaTumora.set(2300)

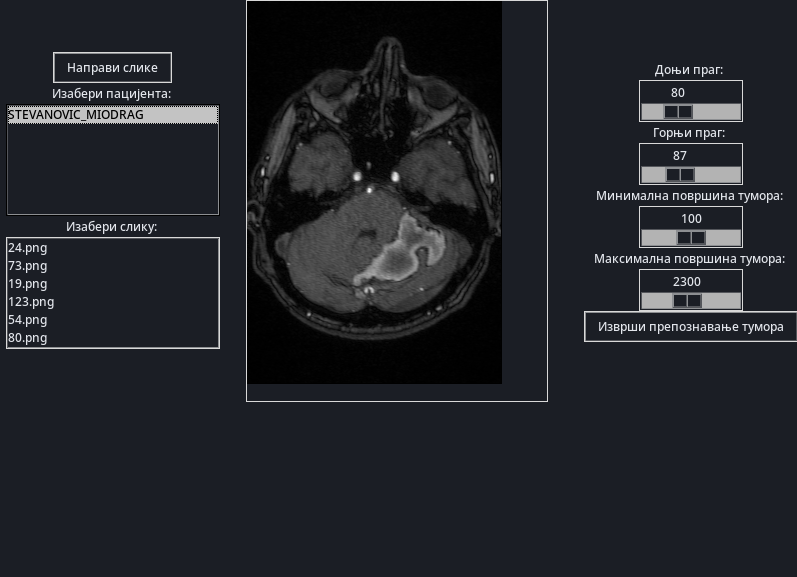
self.\_\_maksimalnaPovrsinaTumora.grid(column=0, row=7)

self.\_\_dugmeIzvrsiSlike = tk.Button(self.\_\_kolona3, text="Изврши препознавање тумора",

command=self.izvrsiPrepoznavanjeHandler)

self.\_\_dugmeIzvrsiSlike.grid(column=0, row=8)

Блок кода 6 – Функција napraviUI



Слика 7 – Кориснички интерфејс

Следећа функција је *napraviBindove*, не прима параметре, и служи за повезивање постојећих исцртаних елемената са одговарајућим функцијама. Конкретно, повезујемо клик миша на оба listBox-a са функцијом *proveraSlika*, и изабир елемената са *pacijentSelected* и *slikaSelected* за *listboxPacijent* и *listboxSlike* респективно.

def napraviBindove(self):

self.\_\_listboxPacijent.bind('<ButtonRelease>', self.proveraSlika)

self.\_\_listboxSlike.bind('<ButtonRelease>', self.proveraSlika)

self.\_\_listboxPacijent.bind('<<ListboxSelect>>', self.pacijentSelected)

self.\_\_listboxSlike.bind('<<ListboxSelect>>', self.slikaSelected)

Блок кода 7 – Функција napraviBindove

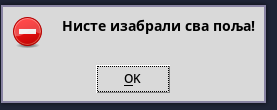
Функција *napraviSlikeHandler* се позива притиском на дугме на ком пише “Направи слике” (*dugmeNapraviSlike* у класи), ова функција је повезана у констрактору. Притиском на дугме, позива се функција *napraviSlike* из *DicomHandler-а*, док у параметрима дајемо *listboxPacijent* да би *DicomHandler* могао да измени садржај истог.

def napraviSlikeHandler(self):

DicomHandler.napraviSlike(self.\_\_listboxPacijent)

Блок кода 8 – Функција napraviSlikeHandler

Функција izvrsiPrepoznavanjeHandler не прима параметре, али позвана је притиском на дугме са текстом “Изврши препознавање тумора”. Ово дешавање је повезано у констрактору класе. Функција прво проверава да ли су сва поља изабрана (конкретно да ли су пацијент и слика изабрани), ако нису упозорава корисника да није изабрао слику. Ова грешка изгледа као на слици 8, и изглед зависи од теме система.



Слика 8 – Грешка у одабиру поља

Ако не буде грешке, функција наставља и прави нову инстанцу класе *Detekcija* са параметрима путање слике која је изабрана. Након овога, користимо функције *detektujTumor*, где параметре учитавамо из одговарајућих поља у корисничком интерфејсу (доњи и горњи праг, минимална и максимална површина тумора). На крају, функција позива *prikaziRezultate*, и самим тим искаче прозор на ком се види резултат препознавања тумора. Код ове функције се види на блоку кода 9.

def izvrsiPrepoznavanjeHandler(self):

if not (self.\_\_listboxSlike.curselection() and self.\_\_listboxPacijent.curselection()):

messagebox.showerror("Грешка!", "Нисте изабрали сва поља!")

return

selectedSlika = self.\_\_listboxSlike.get(self.\_\_listboxSlike.curselection())

selectedPacijent = self.\_\_listboxPacijent.get(self.\_\_listboxPacijent.curselection())

fullPutanja = f"./slike/{selectedPacijent}/{selectedSlika}"

detekcija = Detekcija(fullPutanja)

detekcija.detektujTumor(self.\_\_donjiThresholdRange.get(),

self.\_\_gornjiThresholdRange.get(),

self.\_\_minimalnaPovrsinaTumora.get(),

self.\_\_maksimalnaPovrsinaTumora.get())

detekcija.prikaziRezultate()

Блок кода 9 – Функција izvrsiPrepoznavanjeHandler

Функција pacijentSelected, прима параметар *event*[[4]](#_Литература), и покреће се изабиром пацијента у *listboxPacijent*, ова активност је повезана у функцији *napraviBindove*. Функција проверава који пацијент је изабран, брише све из *listboxSlike*, и уписује одговарајуће слике пацијента читањем одговарајућег директоријума[[7]](#_Литература) слика пацијента. Код функције се налази у блоку кода 10.

def pacijentSelected(self, event):

selected = self.\_\_listboxPacijent.get(self.\_\_listboxPacijent.curselection())

print(f"Изабран пацијент: {selected}")

slike = os.listdir(f"./slike/{selected}")

self.\_\_listboxSlike.delete(0, tk.END)

for slika in slike:

self.\_\_listboxSlike.insert(tk.END, slika)

Блок кода 10 – Функција pacijentSelected

Функција slikaSelected, прима параметар *event*[[4]](#_Литература), позива се изабиром слике на *listboxSlike*, повезано у функцији *napraviBindove*. При позиву, функција учитава изабрану слику, брише претходну ако постоји, и приказује је у другој колони на *canvasSlike* платну. Код функције се налази у блоку кода 11.

def slikaSelected(self, event):

selectedSlika = self.\_\_listboxSlike.get(self.\_\_listboxSlike.curselection())

selectedPacijent = self.\_\_listboxPacijent.get(self.\_\_listboxPacijent.curselection())

fullPutanja = f"./slike/{selectedPacijent}/{selectedSlika}"

print(

f"Изабрана слика: {fullPutanja}")

img = tk.PhotoImage(file=fullPutanja)

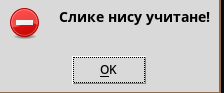
self.\_\_window.img = img

self.\_\_canvasSlike.delete("all")

self.\_\_canvasSlike.create\_image(img.width() / 2, img.height() / 2, anchor="center", image=img)

Блок кода 11 – Функција slikaSelected

Коначно, функција *proveraSlike*, прима параметар *event*[[4]](#_Литература), позива се било каквим кликом на оба *listbox* елемента, проверава да ли постоји било какав пацијент у директоријуму слика[[7]](#_Литература), ако постоји излази из функције, у супротном приказује грешку да слике нису учитане. Грешку можемо видети на слици 9, а код у блоку кода 12.



Слика 9 – Грешка приликом клика на listBox

def proveraSlika(self, event):

if os.path.exists('./slike') and len(os.listdir("./slike")) > 0:

return

messagebox.showerror("Грешка!", "Слике нису учитане!")

Блок кода 12 – Функција proveraSlika

# Фајл: tumor\_prepoznavanje.py

Коначно, долазимо до фајла у ком се налази сама “main функција” програма. Сам фајл садржи једну класу *Main*, и један услов. Класа Main има само констрактор који служи да покрене констрактор класе *UIClass*, и самим тим цео програм.. Стандардно проверавамо да ли је програм позван као програм или је увезен, и ако је покренут, позивамо констрактор *Main* класе, што самим тим покреће програм.

from ui import UIClass

class Main:

def \_\_init\_\_(self):

self.\_\_ui = UIClass()

if \_\_name\_\_ == "\_\_main\_\_":

main = Main()

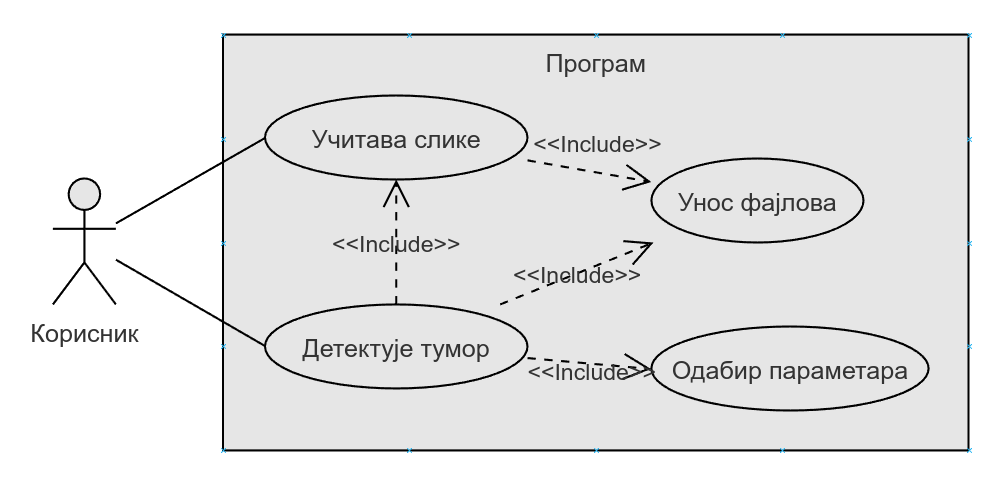
Блок кода 13 – Фајл tumor\_prepoznavanje.py

# UML дијаграми

Након целокупног представљања пројекта, долазимо до UML дијаграма који важе за овај пројекат. Овим начином ближе представљамо пројекат и олакшавамо даље проширење.

# Дијаграм случајева коришћења (use case)

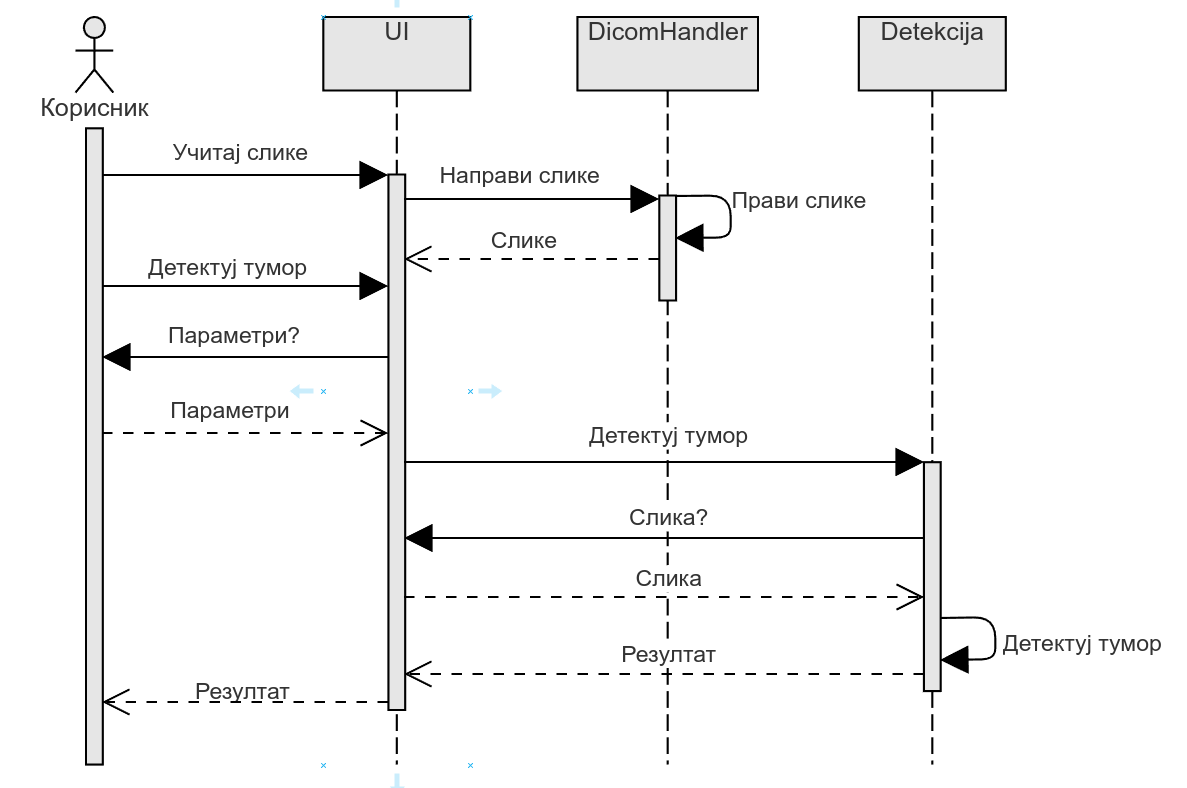
Дијаграм случајева коришћења представља све функције које програм треба да врши, као и све акције које су неопходне да се та главна акција извршила. Код овог програма, главне употребе јесу учитавање медицинских слика из DICOM фајлова, као и детекција тумора на основу изабране слике и унетих параметара. Дијаграм можемо видети на слици 10.



Слика 10 – Дијаграм случајева коришћења

# Дијаграм секвенци

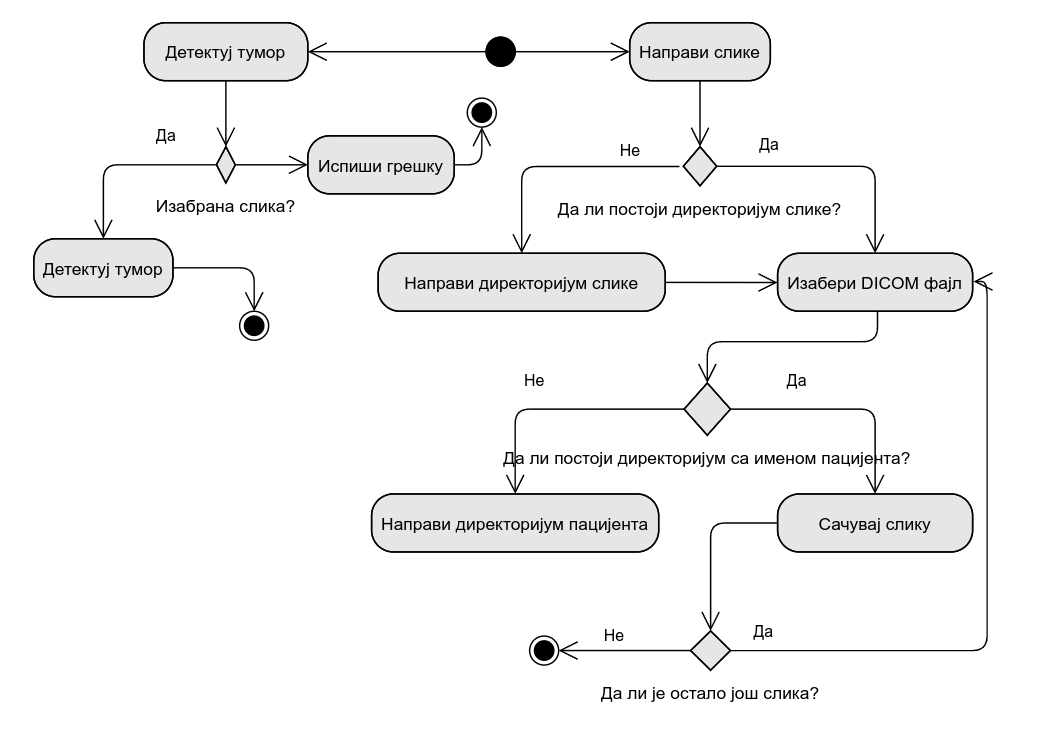
Као други корак, исцртавамо дијаграм секвенци програма. Дијаграм почиње учитавањем слика, и завршава се приказом резултата кориснику. Дијаграм се може видети на слици 11.



Слика 11 – Дијаграм секвенци

# Дијаграм активности

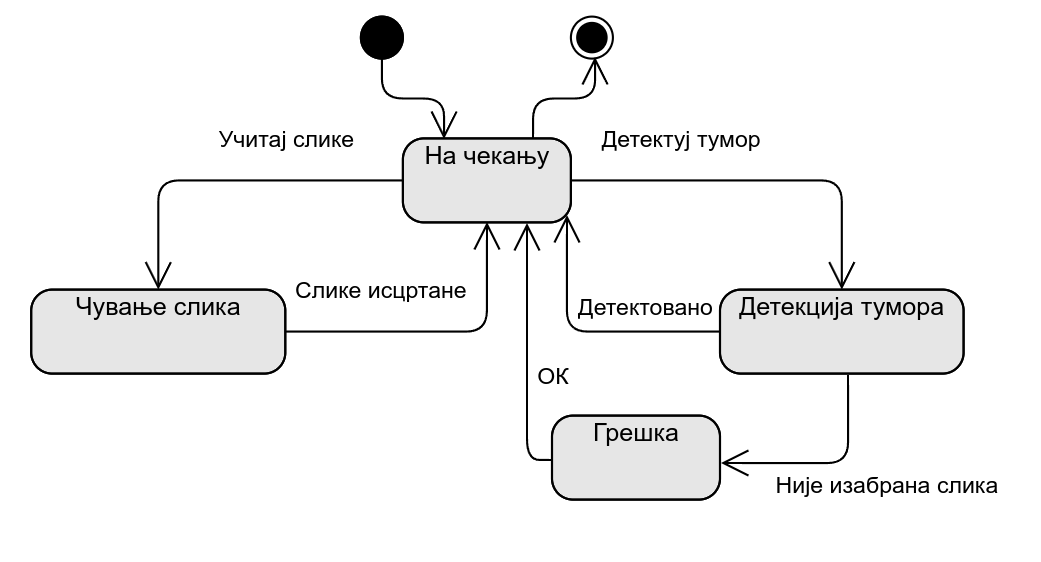
За дијаграм активности, имамо 2 активности – Прављење слика, и детекција тумора. У случају детекције тумора, ако није изабрана слика, програм треба да испише грешку. У случају прављења слика, ако није направљен директоријум слика, треба да га направимо. Исто важи и за директоријум пацијента унутар директоријума слика. Коначно, пролазимо кроз све DICOM фајлове и правимо слике засебно. Дијаграм можемо видети на слици 12.



Слика 12 – Дијаграм активности

# Дијаграм стања

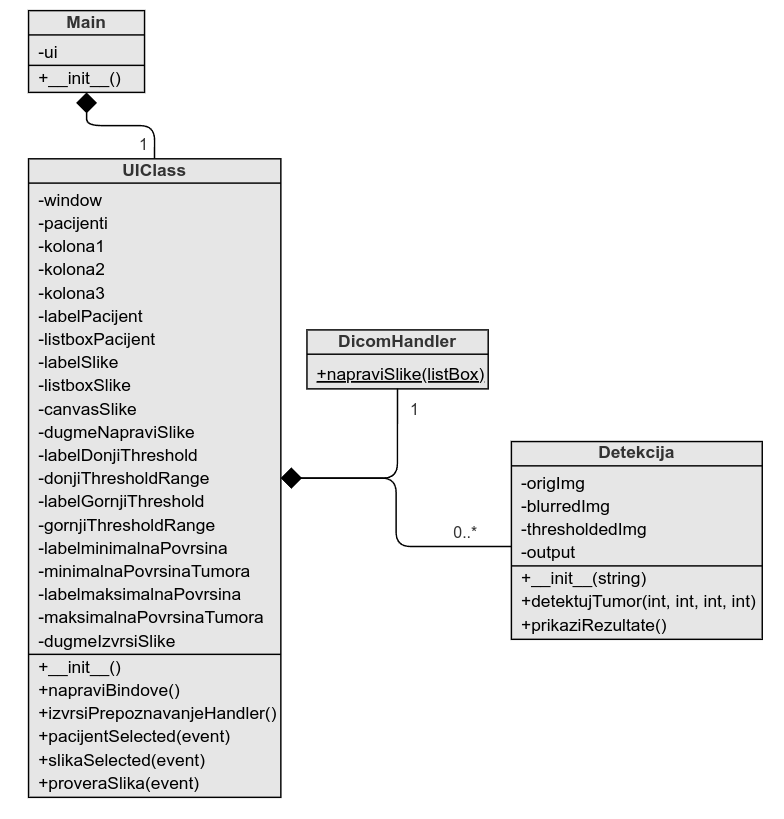
Код дијаграма стања, имамо 4 стања: На чекању, детекција тумора, чување слика и грешка. Дијаграм стања можемо видети на слици 13.



Слика 13 – Дијаграм стања

# Дијаграм класа

Дијаграм класа се састоји из претходно наведених класа Main, UIClass, DicomHandler и Detekcija. Main класа има једну и само једну UIClass, једна класа UIClass има један и само један DicomHandler (класа користи статичку методу унутар DicomHandlera), и класа UIClass такође садржи 0 или више инстанци класа Detekcija. Можемо видети дијаграм класа на слици 14.



Слика 14 – Дијаграм класа

# Литература

[1] <https://www.youtube.com/watch?v=WQK_oOWW5Zo>

[2] <https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html>

[3] <https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html>

[4] <https://wiki.python.org/moin/TkInter>

[5] <https://pydicom.github.io/>

[6] <https://matplotlib.org/stable/tutorials/introductory/images.html>

[7] <https://docs.python.org/3/library/os.html>