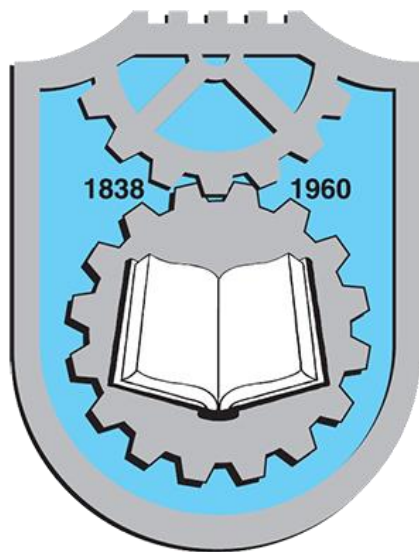


Универзитет у Крагујевцу
Факултет инжењерских наука



Софтверски инжењеринг

Семинарски рад:

Препознавање тумора мозга са 2D снимака DICOM формата

Студент: Алекса Лазаревић

Предметни наставник: др Филиповић Ненад

Крагујевац, Август 2022.

Садржај

1.	Увод	3
2.	Задатак	3
3.	Софтверски алати	3
4.	Структура пројекта и код	4
4.1	Фајл: detekcija.py	4
4.2	Фајл: dicomhandler.py.....	9
4.3	Фајл: ui.py	10
4.4	Фајл: tumor_prepoznavanje.py	17
5.	UML дијаграми.....	17
5.1	Дијаграм случајева коришћења (use case).....	17
5.2	Дијаграм секвенци	18
5.3	Дијаграм активности	19
5.4	Дијаграм стања	19
5.5	Дијаграм класа.....	20
6.	Литература	21

1. Увод

Тумори представљају један од најозбиљнијих и најкомпликованијих проблема медицине 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 смртог исхода. Хитна неурохируршка интервенција је посебно битна за ове типове тумора. Циљ програма би био убрзавање дијагнозе тумора директним читањем са DICOM формата.

2. Задатак

Почетак сваког софтверског пројекта састоји се из јасног састављања задатка, ондносно проблема који сам софтвер решава. Конкретно, 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.

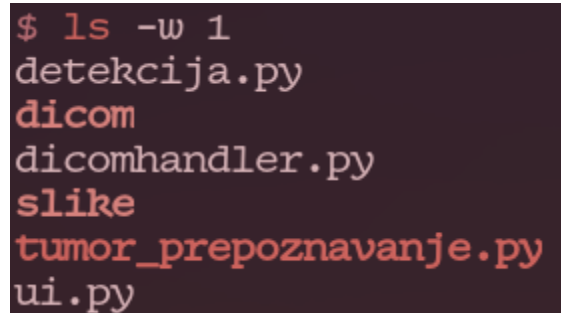
3. Софтверски алати

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-а, 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).

4. 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.



```
$ ls -w 1
detekcija.py
dicom
dicomhandler.py
slike
tumor_prepoznavanje.py
ui.py
```

Слика 1 – директоријум пројекта

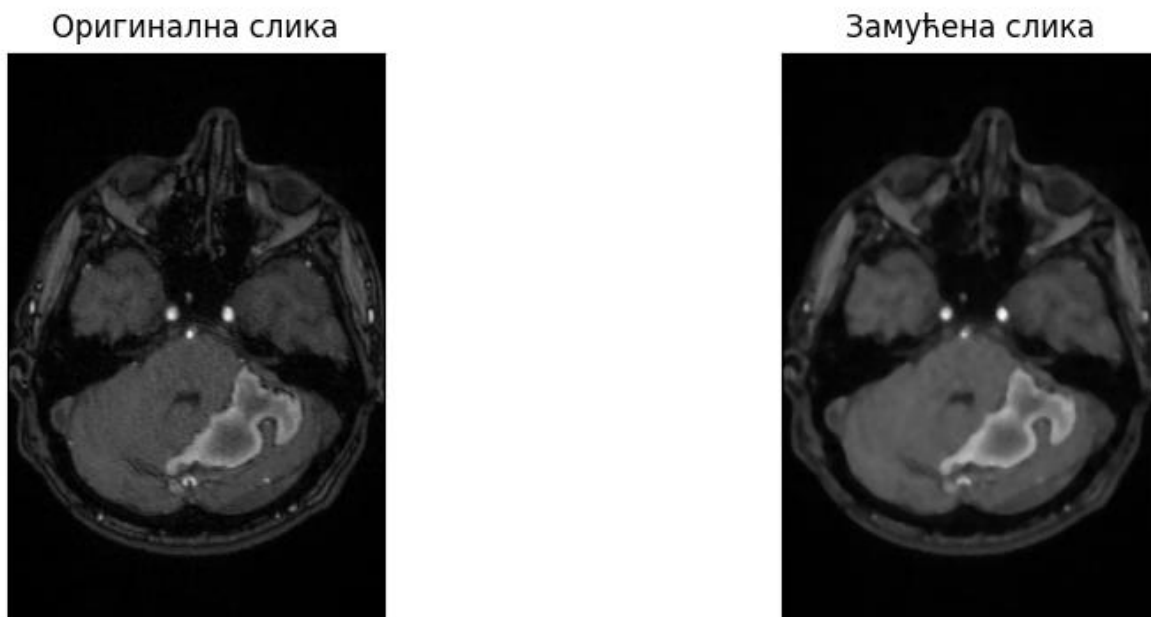
4.1 Фајл: detekcija.py

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 конструктор. Конструктор ове класе прима путању до слике на којој ће детекција тумора да се изврши. Конструктор чита слику и чува је у *ndarray* формату у варијаблу *origImg*. Код конструктора 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 – Конструктор класе *Detekcija*

Поред конструктора, 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 класи. Поред конструктора, 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

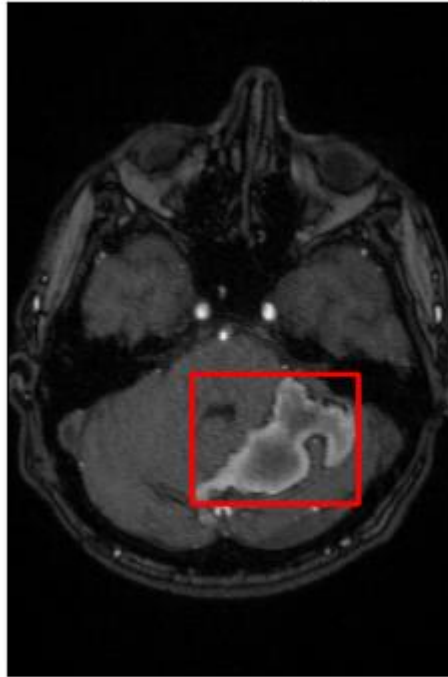
Након замућивања, извршавамо бинарни праг^{[1][2]} на замућену слику (са слике 2), и подешавањем параметара доњег и горњег прага, добијамо резултат као на слици 3. Ову слику чувамо у параметру *tresholddedImg* у класи.



Слика 3 – Слика бинарног прага

Након овог препроцесирања слике, остало нам је да извучемо контуре добијене бинарним прагом, да пронађемо највећу контуру^[3] 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 контуре



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)

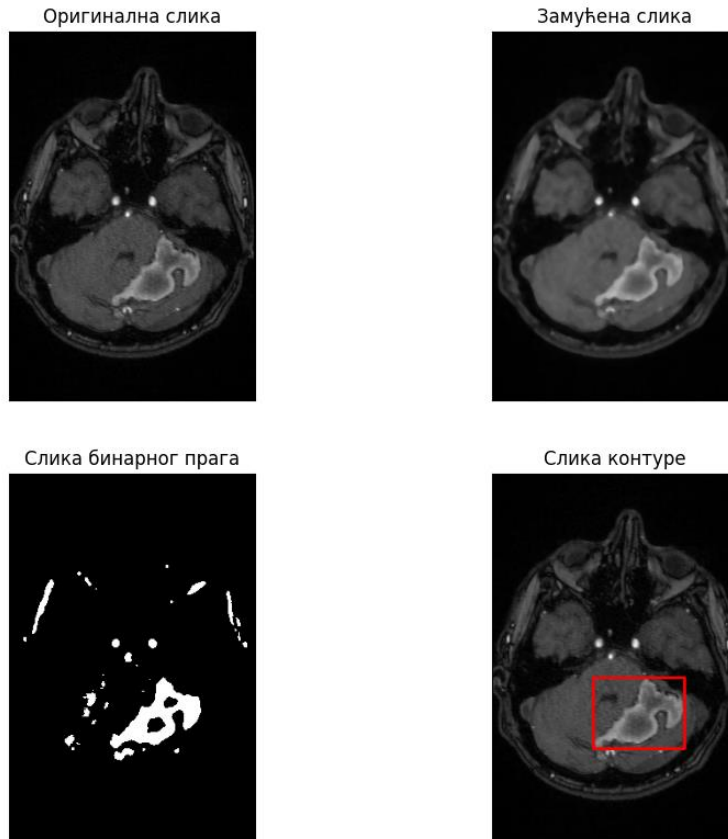
    print(f"Проналазим највећу контуру на слици...")
    contours, hierarchy = cv2.findContours(
        self.__thresholdedImg, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    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] 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([])
```

Блок кода 3 – Функција prikaziRezultate

4.2 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] *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.

```
$ tree slike
slike
├── STEVANOVIĆ_MIODRAG
│   ├── 100.png
│   ├── 101.png
│   ├── 102.png
│   ├── 103.png
│   ├── 104.png
│   ├── 105.png
│   ├── 106.png
│   ├── 107.png
│   ├── 108.png
│   ├── 109.png
│   ├── 10.png
│   └── 110.png
```

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("Проналазимо 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*

4.3 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 конструктор класе. Конструктор 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]. За библиотеку израде корисничког интерфејса изабран је *TkInter*, који долази уз сваку референтну инсталацију *python*-а, дакле у већини случајева ово није неопходно инсталирати. Конструктор се налази у блоку кода 5.

```
def __init__(self):
    self.napraviUI()
    self.napraviBindove()
    self.__window.mainloop()
```

Блок кода 5 – Конструктор 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.__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')

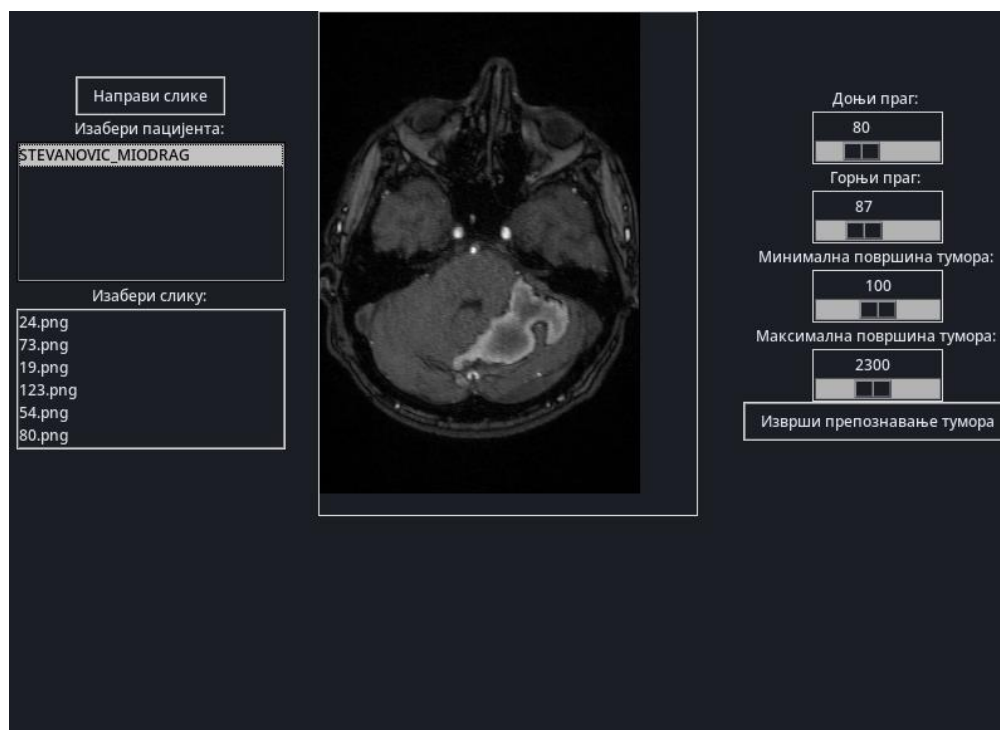
self.__minimalnaPovrsinaTumora = tk.Scale(self.__kolona3, from_=0, to=200, orient='horizontal')
self.__minimalnaPovrsinaTumora.set(100)
self.__minimalnaPovrsinaTumora.grid(column=0, row=5)

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.__dugmeIzvrshiSlike = tk.Button(self.__kolona3, text="Изврши препознавање тумора",
                                     command=self.izvrshiPrepoznavanjeHandler)
self.__dugmeIzvrshiSlike.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-а 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)
```

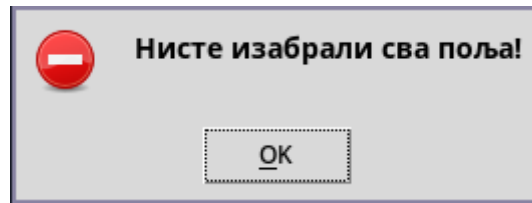
Блок кода 7 – Функција *napraviBindove*

Функција *napraviSlikeHandler* се позива притиском на дугме на ком пише “Направи слике” (*dugmeNapraviSlike* у класи), ова функција је повезана у конструктору. Притиском на дугме, позива се функција *napraviSlike* из *DicomHandler-a*, 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 конструктору 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
    selectedSlike = self.__listboxSlike.get(self.__listboxSlike.curselection())
    selectedPacijent = self.__listboxPacijent.get(self.__listboxPacijent.curselection())
    fullPutanja = f"./slike/{selectedPacijent}/{selectedSlike}"
    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], 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] 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*

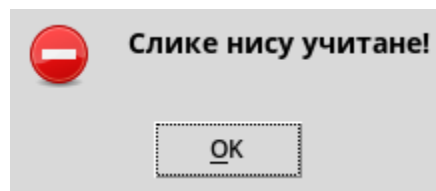
Функција *slikaSelected*, прима параметар *event*^[4], позива се изабиром слике на *listboxSlike*, повезано у функцији *napraviBindove*. При позиву, функција учитава изабрану слику, брише претходну ако постоји, и приказује је у другој колони на *canvasSlike* платну. Код функције се налази у блоку кода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

Блок кода 11 – Функција *slikaSelected*

Коначно, функција *proveraSlike*, прима параметар *event*^[4], позива се било каквим кликом на оба *listbox* елемента, проверава да ли постоји било какав пацијент у директоријуму слика^[7], 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*

4.4 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 конструктор који служи да покрене конструктор класе *UIClass*, и самим тим цео програм.. Стандардно проверавамо да ли је програм позван као програм или је увезен, и ако је покренут, позивамо конструктор *Main* класе, што самим тим покреће програм.

```
from ui import UIClass

class Main:
    def __init__(self):
        self._ui = UIClass()

if __name__ == "__main__":
    main = Main()
```

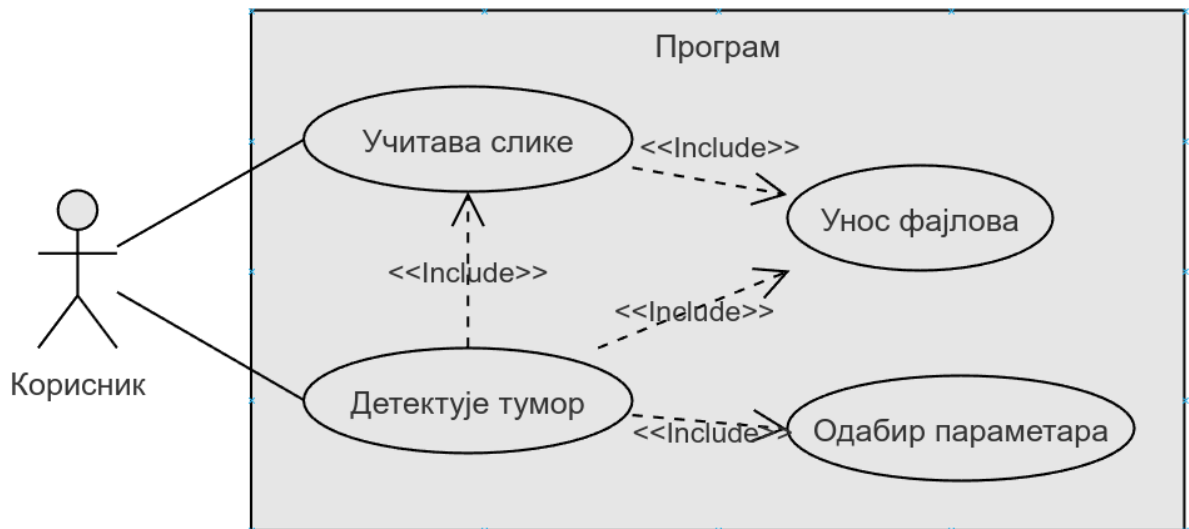
Блок кода 13 – Фајл *tumor_prepoznavanje.py*

5. UML дијаграми

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

5.1 Дијаграм случајева коришћења (use case)

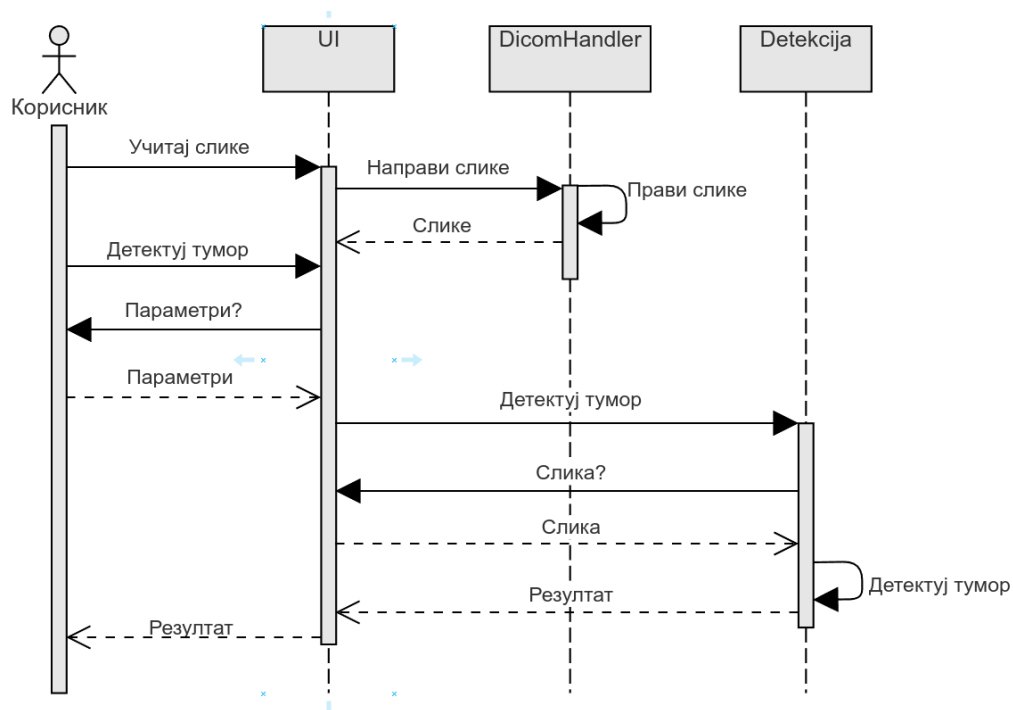
Дијаграм случајева коришћења представља све функције које програм треба да врши, као и све акције које су неопходне да се та главна акција извршила. Код овог програма, главне употребе јесу учитавање медицинских слика из DICOM фајлова, као и детекција тумора на основу изабране слике и унетих параметара.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



Слика 10 – Дијаграм случајева коришћења

5.2 Дијаграм секвенци

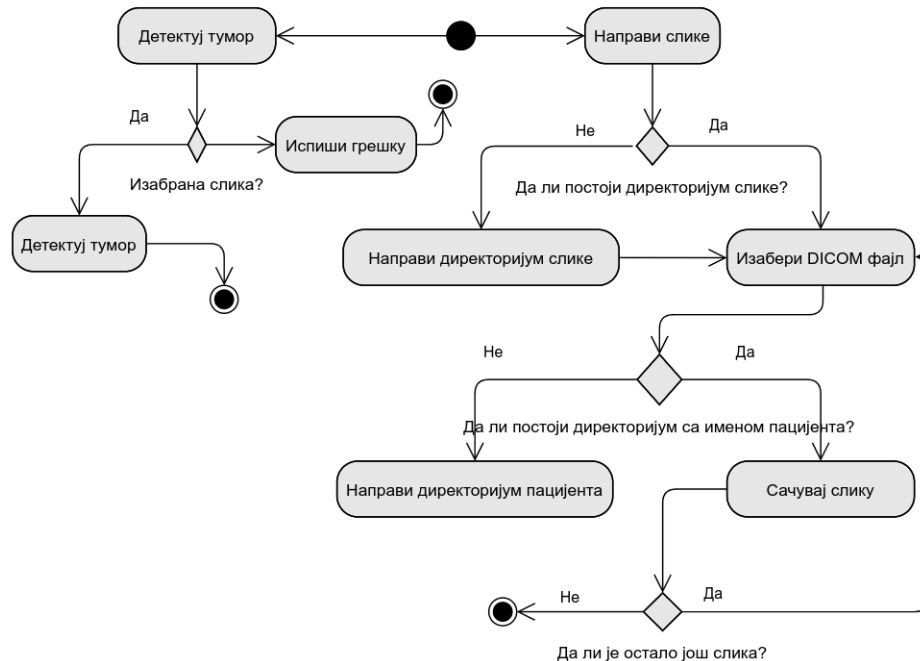
Као други корак, исцртавамо дијаграм секвенци програма. Дијаграм почиње учитавањем слика, и завршава се приказом резултата кориснику. Дијаграм се може видети на слици 11.



Слика 11 – Дијаграм секвенци

5.3 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

5.4 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

5.5 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

6. 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>