**Algorithm Main**

Import🡪 QtCore 🡪 PyQt5

Import 🡪 Generate, Instructor, ResultViewer, Room, Subject, Section 🡪 containers

Import 🡪 Settings, Database, Timetable, ImportExportHandler as ioHandler 🡪 components

Import 🡪 Main 🡪 py\_ui

Import 🡪 json

Import 🡪 gc

Class 🡨 MainWindow(Main.Ui\_MainWindow)

Function 🡨 \_\_init\_\_(self, parent)

super().\_\_init\_\_()

self.parent 🡨 parent

self.setupUi(parent)

self.connectButtons()

self.settings 🡨 Settings.getSettings()

self.loadSettings()

self.handleSettings()

self.drawTrees()

self.tabWidget.currentChanged.connect(self.tabListener)

self.tabWidget.setCurrentIndex(0)

# Connect Main component buttons to respective actions

Function 🡨 connectButtons(self)

self.btnInstrAdd.clicked.connect(lambda self.openInstructor())

self.btnRoomAdd.clicked.connect(lambda self.openRoom())

self.btnSubjAdd.clicked.connect(lambda self.openSubject())

self.btnSecAdd.clicked.connect(lambda self.openSection())

self.btnScenResult.clicked.connect(lambda self.openResult())

self.btnScenGenerate.clicked.connect(lambda self.openGenerate())

self.btnInstrImport.clicked.connect(self.importInstructors)

self.btnRoomImport.clicked.connect(self.importRooms)

self.btnSubjImport.clicked.connect(self.importSubjects)

self.actionSave\_As.triggered.connect(self.saveAs)

self.actionOpen.triggered.connect(self.load)

self.actionSettings.triggered.connect(lambda self.tabWidget.setCurrentIndex(4))

self.actionExit.triggered.connect(exit)

self.actionNew.triggered.connect(lambda self.new())

# Initialize trees and tables

Function 🡨 drawTrees(self)

self.instrTree 🡨 Instructor.Tree(self.treeInstr)

self.roomTree 🡨 Room.Tree(self.treeRoom)

self.subjTree 🡨 Subject.Tree(self.treeSubj)

self.secTree 🡨 Section.Tree(self.treeSec)

# Handle component openings

Function 🡨 openInstructor(self, id<-False)

Instructor.Instructor(id)

self.instrTree.display()

Function 🡨 openRoom(self, id<-False)

Room.Room(id)

self.roomTree.display()

Function 🡨 openSubject(self, id<-False)

Subject.Subject(id)

self.subjTree.display()

Function 🡨 openSection(self, id<-False)

Section.Section(id)

self.secTree.display()

Function 🡨 tabListener(self, index)

self.instrTree.display()

self.roomTree.display()

self.subjTree.display()

self.secTree.display()

IF index == 4

self.checkContents()

Function 🡨 checkContents(self)

conn 🡨 Database.getConnection()

cursor 🡨 conn.cursor()

disabled 🡨 False

cursor.execute('SELECT id FROM rooms LIMIT 1')

IF cursor.fetchone()

disabled 🡨 True

cursor.execute('SELECT id FROM instructors LIMIT 1')

IF cursor.fetchone()

disabled 🡨 True

cursor.execute('SELECT id FROM sections LIMIT 1')

IF cursor.fetchone()

disabled 🡨 True

cursor.execute('SELECT id FROM subjects LIMIT 1')

IF cursor.fetchone()

disabled 🡨 True

self.timeStarting.setDisabled(disabled)

self.timeEnding.setDisabled(disabled)

self.btnScenGenerate.setDisabled(not disabled)

conn.close()

Function 🡨 openResult(self)

ResultViewer.ResultViewer()

Function 🡨 openGenerate(self)

gc.collect()

result 🡨 Generate.Generate()

IF not len(result.topChromosomes)

return False

self.openResult()

Function 🡨 importInstructors(self)

instructors 🡨 ioHandler.getCSVFile('instructors')

IF instructors

instructors.pop(0)

instructors.pop(0)

blankSchedule 🡨 json.dumps(Timetable.generateRawTable())

for instructor in instructors

Instructor.Instructor.insertInstructor([instructor[0], float(instructor[1]), blankSchedule])

self.tabListener(0)

Function 🡨 importRooms(self)

rooms 🡨 ioHandler.getCSVFile('rooms')

IF rooms

rooms.pop(0)

rooms.pop(0)

blankSchedule 🡨 json.dumps(Timetable.generateRawTable())

for room in rooms

Room.Room.insertRoom([room[0], blankSchedule, room[1]])

self.tabListener(1)

Function 🡨 importSubjects(self)

subjects 🡨 ioHandler.getCSVFile('subjects')

IF subjects

subjects.pop(0)

subjects.pop(0)

for subject in subjects

Subject.Subject.insertSubject(

[subject[1], float(subject[3]), subject[0], '', json.dumps([]), int(subject[4]), subject[2]])

self.tabListener(2)

Function 🡨 saveAs(self)

ioHandler.saveAs()

Function 🡨 load(self)

ioHandler.load()

self.tabWidget.setCurrentIndex(0)

self.tabListener(0)

Function 🡨 loadSettings(self)

self.timeStarting.setTime(QtCore.QTime(int(self.settings['starting\_time'] / 2), 0))

self.timeEnding.setTime(QtCore.QTime(int(self.settings['ending\_time'] / 2) + 1, 0))

IF self.settings['lunchbreak']

self.radioLunchYes.setChecked(True)

ELSE

self.radioLunchNo.setChecked(True)

self.editMinPop.setValue(self.settings['minimum\_population'])

self.editMaxPop.setValue(self.settings['maximum\_population'])

self.editMaxGen.setValue(self.settings['maximum\_generations'])

self.editMaxCreation.setValue(self.settings['generation\_tolerance'])

self.editMut.setValue(self.settings['mutation\_rate\_adjustment\_trigger'])

self.editMaxFit.setValue(self.settings['maximum\_fitness'])

self.editElite.setValue(int(self.settings['elite\_percent'] \* 100))

self.editDev.setValue(self.settings['deviation\_tolerance'])

self.matrix 🡨 matrix 🡨 self.settings['evaluation\_matrix']

self.editSbj.setValue(matrix['subject\_placement'])

self.editLun.setValue(matrix['lunch\_break'])

self.editSec.setValue(matrix['student\_rest'])

self.editIdle.setValue(matrix['idle\_time'])

self.editInstrRest.setValue(matrix['instructor\_rest'])

self.editInstrLoad.setValue(matrix['instructor\_load'])

self.editMeet.setValue(matrix['meeting\_pattern'])

self.matrixSum 🡨 sum(matrix.values())

self.lblTotal.setText('Total {}%'.format(self.matrixSum))

# Handle Settings

Function 🡨 handleSettings(self)

self.timeStarting.timeChanged.connect(self.handleStartingTime)

self.timeEnding.timeChanged.connect(self.handleEndingTime)

self.radioLunchYes.toggled.connect(lambda state self.updateSettings('lunchbreak', state))

self.editMinPop.valueChanged.connect(self.handleMinPop)

self.editMaxPop.valueChanged.connect(self.handleMaxPop)

self.editMaxGen.valueChanged.connect(lambda value self.updateSettings('maximum\_generations', value))

self.editMaxCreation.valueChanged.connect(lambda value self.updateSettings('generation\_tolerance', value))

self.editMut.valueChanged.connect(

lambda value self.updateSettings('mutation\_rate\_adjustment\_trigger', round(value, 2)))

self.editMaxFit.valueChanged.connect(lambda value self.updateSettings('maximum\_fitness', value))

self.editElite.valueChanged.connect(lambda value self.updateSettings('elite\_percent', round(value / 100, 2)))

self.editDev.valueChanged.connect(lambda value self.updateSettings('deviation\_tolerance', value))

self.editSbj.valueChanged.connect(lambda value self.handleMatrix('subject\_placement', value, self.editSbj))

self.editLun.valueChanged.connect(lambda value self.handleMatrix('lunch\_break', value, self.editLun))

self.editSec.valueChanged.connect(lambda value self.handleMatrix('student\_rest', value, self.editSec))

self.editIdle.valueChanged.connect(lambda value self.handleMatrix('idle\_time', value, self.editIdle))

self.editInstrRest.valueChanged.connect(

lambda value self.handleMatrix('instructor\_rest', value, self.editInstrRest))

self.editInstrLoad.valueChanged.connect(

lambda value self.handleMatrix('instructor\_load', value, self.editInstrLoad))

self.editMeet.valueChanged.connect(lambda value self.handleMatrix('meeting\_pattern', value, self.editMeet))

Function handleStartingTime(self, time)

IF time.hour() \* 2 >= self.settings['ending\_time']

self.timeStarting.setTime(QtCore.QTime(int(self.settings['starting\_time'] / 2), 0))

ELSE

self.updateSettings('starting\_time', time.hour() \* 2)

Function 🡨 handleEndingTime(self, time)

IF (time.hour() \* 2) - 1 <= self.settings['starting\_time']

self.timeEnding.setTime(QtCore.QTime(int(self.settings['ending\_time'] / 2) + 1, 0))

ELSE

self.updateSettings('ending\_time', (time.hour() \* 2) - 1)

Function 🡨 handleMinPop(self, value)

IF value > self.settings['maximum\_population']

self.editMinPop.setValue(self.settings['minimum\_population'])

ELSE

self.updateSettings('minimum\_population', value)

Function 🡨 handleMaxPop(self, value)

IF value < self.settings['minimum\_population']

self.editMaxPop.setValue(self.settings['maximum\_population'])

ELSE

self.updateSettings('maximum\_population', value)

Function handleMatrix(self, key, value, obj)

dIFference 🡨 self.matrix[key] - value

IF self.matrixSum - difference > 100

obj.setValue(self.matrix[key])

ELSE

self.updateSettings('evaluation\_matrix', value, key)

self.matrixSum 🡨 sum(self.settings['evaluation\_matrix'].values())

self.matrix 🡨 self.settings['evaluation\_matrix']

self.lblTotal.setText('Total {}%'.format(self.matrixSum))

Function 🡨 updateSettings(self, key, value, secondKey<-False)

Settings.setSettings(key, value, secondKey)

self.settings 🡨 Settings.getSettings()

Function 🡨 new(self)

ioHandler.removeTables()

Database.setup()

self.tabListener(0)