**Algorithm: Generate**

From PyQt5 🡨 Import 🡪 QtCore, QtWidgets

From components 🡨 Import 🡪 Database as db, ResourceTracker, ScheduleParser, ScenarioComposer, GeneticAlgorithm

From py\_ui 🡨 Import 🡪 Generate as Parent

From sqlite3 🡨 Import 🡪 Binary

From numpy 🡨 Import 🡪 mean

Import 🡪 pickle

Import 🡪 copy

Class 🡨 Generate

Function 🡨 \_\_init\_\_(self):

self.totalResource 🡨 {

'cpu': [],

'memory': []

}

self.tick 🡨 0

self.data 🡨 {

'results': [],

'rooms': [],

'instructors': [],

'sections': [],

'sharings': [],

'subjects': []

}

self.topChromosomes 🡨 []

self.meta 🡨 []

self.preview 🡨 True

self.sectionKeys 🡨 []

composer 🡨 ScenarioComposer.ScenarioComposer()

composer 🡨 composer.getScenarioData()

self.data.update(composer)

self.dialog 🡨 dialog 🡨 QtWidgets.QDialog(parent=None)

# Initialize custom dialog

self.parent 🡨 parent 🡨 Parent.Ui\_Dialog()

# Add parent to custom dialog

parent.setupUi(dialog)

dialog.setWindowFlags(QtCore.Qt.Window | QtCore.Qt.WindowTitleHint | QtCore.Qt.CustomizeWindowHint)

self.time 🡨 QtCore.QTime(0, 0)

self.timer 🡨 QtCore.QTimer()

self.timer.timeout.connect(self.updateTime)

self.timer.start(1000)

self.running 🡨 True

self.table 🡨 parent.tableSchedule

parent.btnPause.clicked.connect(self.togglePause)

parent.btnStop.clicked.connect(self.stopOperation)

parent.chkPreview.clicked.connect(self.togglePreview)

parent.cmbSection.clear()

FOR section, details IN self.data['sections'].items():

self.sectionKeys.append(section)

parent.cmbSection.addItem(details[0])

END FOR loop

parent.cmbSection.currentIndexChanged.connect(self.changePreview)

self.startWorkers()

dialog.exec\_()

Function 🡨 togglePreview(self, state):

self.preview 🡨 not state

Function 🡨 togglePause(self):

self.toggleState()

self.parent.btnPause.setText('Pause Generation' if self.running else 'Resume Generation')

Function 🡨 toggleState(self, state=None):

self.running 🡨 (not self.running) if state is None else state

self.resourceWorker.running 🡨 self.running

self.geneticAlgorithm.running 🡨 self.running

Function 🡨 startWorkers(self):

self.resourceWorker 🡨 ResourceTrackerWorker()

self.resourceWorker.signal.connect(self.updateResource)

self.resourceWorker.start()

self.geneticAlgorithm = GeneticAlgorithm.GeneticAlgorithm(self.data)

self.geneticAlgorithm.statusSignal.connect(self.updateStatus)

self.geneticAlgorithm.detailsSignal.connect(self.updateDetails)

self.geneticAlgorithm.dataSignal.connect(self.updateView)

self.geneticAlgorithm.operationSignal.connect(self.updateOperation)

self.geneticAlgorithm.start()

Function 🡨 updateStatus(self, status):

self.parent.lblStatus.setText('Status: {}'.format(status))

Function 🡨 updateDetails(self, details):

self.parent.boxGen.setTitle('Generation #{}'.format(details[0]))

self.parent.lblPopulation.setText('Population: {}'.format(details[1]))

self.parent.lblMutation.setText('Mutation Rate: {}%'.format(details[2]))

self.parent.lblFitness.setText('Average Fitness: {}%'.format(details[3]))

self.parent.lblPreviousFitness.setText('Previous Average Fitness: {}%'.format(details[4]))

self.parent.lblHighestFitness.setText('Highest Fitness: {}%'.format(details[5]))

self.parent.lblLowestFitness.setText('Lowest Fitness: {}%'.format(details[6]))

Function 🡨 updateView(self, chromosomes):

chromosomes.reverse()

self.topChromosomes 🡨 copy.deepcopy(chromosomes)

self.changePreview(self.parent.cmbSection.currentIndex())

Function 🡨 changePreview(self, index):

data 🡨 []

IF NOT len(self.topChromosomes) OR NOT self.preview:

Return 🡨 False

sections 🡨 self.topChromosomes[0][0].data['sections']

rawData 🡨 self.data

subjects 🡨 sections[self.sectionKeys[index]]['details']

FOR subject, details IN subjects.items():

IF NOT len(details):

Continue

END IF

instructor 🡨 '' if not details[1] else rawData['instructors'][details[1]][0]

data.append({'color': None, 'text': '{} \n {} \n

{}'.format(rawData['subjects'][subject][0], rawData['rooms'][details[0]][0], instructor),'instances': [[day, details[3], details[3] + details[4]] for day in details[2]]})

END FOR LOOP

self.loadTable(data)

Function 🡨 loadTable(self, data=[]):

self.table.reset()

self.table.clearSpans()

ScheduleParser.ScheduleParser(self.table, data)

Function 🡨 updateOperation(self, type):

IF type == 1:

self.stopOperation()

Function 🡨 updateTime(self):

self.time 🡨 self.time.addSecs(1)

self.parent.lblTime.setText('Elapsed Time: {}'.format(self.time.toString('hh:mm:ss')))

Function 🡨 stopOperation(self):

self.toggleState(False)

self.resourceWorker.terminate()

self.resourceWorker.runThread 🡨 False

self.geneticAlgorithm.terminate()

self.timer.stop()

IF len(self.topChromosomes):

self.parent.btnStop.setText('View Result')

self.parent.btnStop.clicked.disconnect(self.stopOperation)

self.parent.btnStop.clicked.connect(self.dialog.close)

self.parent.lblCPU.setText('CPU Usage: Stopped')

self.parent.lblMemory.setText('Memory Usage: Stopped')

self.parent.lblStatus.setText('Status: Stopped')

self.totalResource['cpu'] = mean(self.totalResource['cpu'])

self.totalResource['memory'] = mean(self.totalResource['memory'])

self.meta 🡨 [[chromosome[1], chromosome[0].fitnessDetails] FOR chromosome IN self.topChromosomes]

conn 🡨 db.getConnection()

cursor 🡨 conn.cursor()

cursor.execute('INSERT INTO results (content) VALUES (?)', [Binary(pickle.dumps({'data': [chromosome[0].data for chromosome in self.topChromosomes], 'meta': self.meta, 'time': self.time.toString('hh:mm:ss'), 'resource': self.totalResource, 'rawData': self.data}, pickle.HIGHEST\_PROTOCOL))])

conn.commit()

conn.close()

END IF

ELSE:

self.dialog.close()

Function 🡨 updateResource(self, resource):

self.tick 🡨 self.tick + 1

IF self.tick == 3:

self.tick 🡨 0

ELSE:

self.totalResource['cpu'].append(resource[0])

self.totalResource['memory'].append(resource[1][1])

self.parent.lblCPU.setText('CPU Usage: {}%'.format(resource[0]))

self.parent.lblMemory.setText('Memory Usage: {}% - {} MB'.format(resource[1][0], resource[1][1]))

Class 🡨 ResourceTrackerWorker(QtCore.QThread):

signal 🡨 QtCore.pyqtSignal(object)

running 🡨 True

runThread 🡨 True

Function 🡨 \_\_init\_\_(self):

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

Function 🡨 \_\_del\_\_(self):

self.wait()

Function 🡨 run(self):

WHILE (self.runThread):

self.sleep(1)

IF self.running IS True:

cpu 🡨 ResourceTracker.getCPUUsage()

memory 🡨 ResourceTracker.getMemoryUsage()

memory 🡨 [ResourceTracker.getMemoryPercentage(memory), ResourceTracker.byteToMegabyte(memory[0])]

self.signal.emit([cpu, memory])

END IF

END WHILE LOOP

Return 🡨 True