**Algorithm: ResultViewer**

from PyQt5 🡨 import 🡪 QtWidgets

from components 🡨 import 🡪 Settings, Database as db, ScheduleParser

from py\_ui 🡨 import 🡪 Result as Parent

import 🡪 pickle

import 🡪 json

import 🡪 csv

import 🡪 copy

Class 🡨 ResultViewer

Function 🡨 \_\_init\_\_(self):

self.dialog 🡨 dialog 🡨 QtWidgets.QDialog()

# Initialize custom dialog

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

# Add parent to custom dialog

parent.setupUi(dialog)

self.table 🡨 self.parent.tableResult

self.run 🡨 True

self.settings 🡨 Settings.getSettings()

self.result 🡨 { 'data': [] }

self.getLastResult()

IF self.run:

self.parseResultDetails()

self.connectWidgets()

self.updateTable(0)

dialog.exec\_()

END IF

END Function

Function 🡨 getLastResult(self):

conn 🡨 db.getConnection()

cursor 🡨 conn.cursor()

cursor.execute('SELECT content FROM results WHERE id = (SELECT MAX(id) FROM results)')

result 🡨 cursor.fetchone()

conn.close()

IF result:

THEN self.result = pickle.loads(result[0])

ELSE:

messageBox 🡨 QtWidgets.QMessageBox()

messageBox.setWindowTitle('No Data')

messageBox.setIcon(QtWidgets.QMessageBox.Information)

messageBox.setText('You haven\'t generated a solution yet!')

messageBox.setStandardButtons(QtWidgets.QMessageBox.Ok)

messageBox.exec\_()

self.run 🡨 False

END ELSE

END Function

Function 🡨 parseResultDetails(self):

IF NOT len(self.result['data']):

THEN Return 🡨 False

END IF

result 🡨 self.result

self.rawData 🡨 copy.deepcopy(result['rawData'])

self.parent.lblTime.setText('Generation Time: {}'.format(result['time']))

self.parent.lblCPU.setText('Average CPU Usage: {}%'.format(round(result['resource']['cpu']), 2))

self.parent.lblMemory.setText('Average Mem Usage: {} MB'.format(round(result['resource']['memory']), 2))

self.updateEntries(0)

self.updateDetails(0)

Function 🡨 connectWidgets(self):

self.parent.cmbChromosome.currentIndexChanged.connect(self.updateDetails)

self.parent.cmbCategory.currentIndexChanged.connect(self.updateEntries)

self.parent.cmbEntry.currentIndexChanged.connect(self.updateTable)

self.parent.btnExport.clicked.connect(self.export)

Function 🡨 updateDetails(self, index):

parent 🡨 self.parent

meta 🡨 self.result['meta'][index]

parent.lblFit.setText('Total Fitness: {}%'.format(meta[0]))

parent.lblSbj.setText('Subject Placement: {}%'.format(meta[1][0]))

parent.lblSecRest.setText('Section Rest: {}%'.format(meta[1][2]))

parent.lblSecIdle.setText('Section Idle Time: {}%'.format(meta[1][4]))

parent.lblInstrRest.setText('Instructor Rest: {}%'.format(meta[1][3]))

parent.lblInstrLoad.setText('Instructor Load: {}%'.format(meta[1][6]))

parent.lblLunch.setText('Lunch Break: {}%'.format(meta[1][1]))

parent.lblMeet.setText('Meeting Pattern: {}%'.format(meta[1][5]))

parent.cmbCategory.setCurrentIndex(0)

parent.cmbEntry.setCurrentIndex(0)

self.updateEntries(0)

self.updateTable(0)

Function 🡨 updateEntries(self, index):

IF index == 0:

THEN key <- 'sections'

ELSE IF index == 1:

THEN key 🡨 'rooms'

ELSE key 🡨 'instructors'

self.entryKeys = []

self.changingKeys = True

self.parent.cmbEntry.clear()

FOR id, entry in self.rawData[key].items():

self.entryKeys.append(id)

self.parent.cmbEntry.addItem(entry[0])

END FOR loop

self.changingKeys 🡨 False

self.updateTable(self.parent.cmbEntry.currentIndex())

Function 🡨 updateTable(self, index):

IF self.changingKeys:

THEN Return 🡨 False

chromosome 🡨 self.result['data'][self.parent.cmbChromosome.currentIndex()]

category 🡨 self.parent.cmbCategory.currentIndex()

# {secId: {'details': {sbjId: [roomId, instructorId, [day/s], startingTS, length]}}}

sections 🡨 chromosome['sections']

rawData 🡨 self.rawData

data 🡨 []

# Section

IF category == 0:

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

FOR subject, details IN subjects.items():

IF NOT len(details):

THEN continue

instructor 🡨 '' IF NOT details[1] ELSE rawData['instructors'][details[1]][0]

data.append({'color': None, 'text': '{} \n {} \n {}'.format(rawData['subjects'][subject][2], rawData['rooms'][details[0]][0], instructor), 'instances': [[day, details[3], details[3] + details[4]] for day in details[2]]})

END FOR loop

END IF

# Room

ELSE IF category == 1:

FOR section, details in sections.items():

FOR subject, subjectDetail in details['details'].items():

IF not len(subjectDetail):

THEN continue

IF subjectDetail[0] != self.entryKeys[index]:

THEN continue

instructor = '' IF not subjectDetail[1] ELSE rawData['instructors'][subjectDetail[1]][0]

data.append({'color': None, 'text': '{} \n {} \n {}'.format(rawData['subjects'][subject][2], rawData['sections'][section][0], instructor),

'instances': [[day, subjectDetail[3], subjectDetail[3] + subjectDetail[4]] FOR day in

subjectDetail[2]]})

END both FOR loop

END ELSE IF

# Instructor

ELSE:

FOR section, details in sections.items():

FOR subject, subjectDetail in details['details'].items():

IF not len(subjectDetail):

THEN continue

IF subjectDetail[1] != self.entryKeys[index]:

THEN continue

data.append({'color': None, 'text': '{} \n {} \n {}'.format(rawData['subjects'][subject][2], rawData['rooms'][subjectDetail[0]][0], rawData['sections'][section][0]), 'instances': [[day, subjectDetail[3], subjectDetail[3] + subjectDetail[4]] FOR day in subjectDetail[2]]})

END FOR loop

END ELSE

self.loadTable(data)

END Function

Function 🡨 loadTable(self, data=[]):

self.table.reset()

self.table.clearSpans()

ScheduleParser.ScheduleParser(self.table, data)

Function 🡨 export(self):

directory 🡨 QtWidgets.QFileDialog().getExistingDirectory(None, 'Select Directory for Export')

IF NOT directory:

THEN Return 🡨 False

WITH open('timeslots.json') as json\_file:

timeslots 🡨 json.load(json\_file)['timeslots']

CLOSE json\_file

fieldnames 🡨 ['Time', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']

rawData 🡨 self.rawData

chromosome 🡨 self.result['data'][self.parent.cmbChromosome.currentIndex()]

# Create schedule for sections

WITH open('{}/sections\_schedule.csv'.format(directory), 'w', newline='') as file:

writer 🡨 csv.writer(file, dialect='excel')

FOR section, subjects in chromosome['sections'].items():

writer.writerow([self.rawData['sections'][section][0]])

writer.writerow(fieldnames)

schedule 🡨 [['' FOR j IN range(6)] FOR i IN

range(self.settings['ending\_time'] - self.settings['starting\_time'] + 1)]

FOR subject, details IN subjects['details'].items():

IF NOT len(details):

THEN continue

instructor 🡨 '' IF NOT details[1] else rawData['instructors'][details[1]][0]

FOR timeslot IN range(details[3], details[3] + details[4]):

FOR day IN details[2]:

schedule[timeslot][day] 🡨 '{} - {} - {}'.format(rawData['subjects'][subject][2],

rawData['rooms'][details[0]][0], instructor)

END FOR loops

END FOR loop

FOR timeslot IN range(self.settings['starting\_time'], self.settings['ending\_time'] + 1):

writer.writerow([timeslots[timeslot], \*schedule[timeslot - self.settings['starting\_time']]])

END FOR loop

writer.writerow([''])

#Create schedule FOR instructors

WITH open('{}/instructors\_schedule.csv'.format(directory), 'w', newline='') as file:

writer 🡨 csv.writer(file, dialect='excel')

FOR instructor IN rawData['instructors'].keys():

writer.writerow([rawData['instructors'][instructor][0]])

writer.writerow(fieldnames)

schedule = [['' FOR j IN range(6)] FOR i IN

range(self.settings['ending\_time'] - self.settings['starting\_time'] + 1)]

FOR section, subjects IN chromosome['sections'].items():

FOR subject, details IN subjects['details'].items():

IF NOT len(details) or details[1] != instructor:

continue

FOR timeslot IN range(details[3], details[3] + details[4]):

FOR day IN details[2]:

schedule[timeslot][day] = '{} - {} - {}'.format(rawData['subjects'][subject][2], rawData['rooms'][details[0]][0],

rawData['sections'][section][0])

FOR timeslot IN range(self.settings['starting\_time'], self.settings['ending\_time'] + 1):

writer.writerow([timeslots[timeslot], \*schedule[timeslot - self.settings['starting\_time']]])

writer.writerow([''])