Escuela Universitaria Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

ESCUELA UNIVERSITARIA POLITÉCNICA

DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)

ANEXOS

DISEÑO DE SOFTWARE PARA ORGANIZACIÓN Y GESTIÓN ADAPTATIVA DE PROYECTOS

SOFTWARE DESIGN FOR ADAPTIVE PROJECT ORGANIZATION AND MANAGEMENT

425.17.42

Autor: Javier Martínez Lahoz
Tutor: Tomás Cortés Arcos
Fecha: Septiembre 2019

Índice de contenido

1. Código de la aplicación de demostración

1.1 Controlador (*Application.py*)

```
from Library import Worker, Duty, abilitiesLevels

from tkinter import Tk
from Graphics import fillMainWindow
from Graphics import WorkerWindow, listWorkersWindow, DutyWindow, listDutiesWindow
from Graphics import modifWorkerWindow, modifDutyWindow, abilityClass
from Graphics import prevPostDutiesWindow
from Graphics import ganntClass, pertClass
from Graphics import changeDB

import sqlite3
from DbInterface import checkDBTables, loadDbWorkers, loadDbDuties
from DbInterface import storeWorkers, storeDuties
from DbInterface import importAbilities


database = None
connectionDB = None
cursorDB = None

workersCount = 0
dutiesCount = 0

workersList = []
dutiesList = []
knowledgeSet = set()

startDuty = 0
endDuty = 0
mainWindow = None

abilitiesSet = 0
abilitiesValues = 0

"""
    MÉTODO DE INICIO Y VENTANA PRINCIPAL
"""
def main():
    global workersCount
    global workersList
    global mainWindow

    addProjectMilestones()
    loadAbilities()

    mainWindow = Tk()
    fillMainWindow(mainWindow, addWorker, addDuty, listWorkers, listDuties, modifyWorker, modifyDuty,
            prevPostDuties, orgDuties, distrDuties, ganntFcn, pertFcn,
            database, changeProyect, saveProyect)
    mainWindow.mainloop()

    return

"""
    MÉTODO DE AÑADIR TRABAJADOR
"""
def addWorker():
    global workersCount
    returns = []

    workerWindow = Tk()
    workerWindowObject = WorkerWindow(workerWindow, knowledgeSet, abilitiesSet, abilitiesEvaluation, returns)
    workerWindow.mainloop()

    workerWindow.destroy()

    if returns[0] is not '':
        workersList.append(Worker(workersCount, returns[0], returns[1]))
        for worker in workersList:
            if worker.id == workersCount:
                for know in returns[2]:
                    if len(know) > 2:
                        knowledgeSet.add(know)
                        worker.addKnowledge(know)
                for ab in returns[3]:
                    worker.updateAbilities(ab[0], ab[1])
                break
        workersCount = workersCount + 1

    del workerWindowObject
    return

"""
        MÉTODO DE AÑADIR TAREA
"""
def addDuty():
    global dutiesCount

    returns = []

    dutyWindow = Tk()
    dutyWindowObject = DutyWindow(dutyWindow, knowledgeSet, abilitiesSet, abilitiesLevels, returns)
    dutyWindow.mainloop()

    dutyWindow.destroy()
    if returns[0] is not '':
        dutiesList.append(Duty(dutiesCount, returns[0], int(returns[1])))
        for duty in dutiesList:
            if duty.id == dutiesCount:
                for req in returns[2]:
                    if len(req) > 2:
                        knowledgeSet.add(req)
                        duty.addRequirements(req)
                for ab in returns[3]:
                    duty.updateAbilities(ab[0], ab[1])
                break
        dutiesCount = dutiesCount + 1

    del dutyWindowObject

"""
        MÉTODO DE LISTAR TRABAJADORES
"""
def listWorkers():
    global workersList

    lstWorkerWindow = Tk()
    listWorkersWindow(lstWorkerWindow, workersList)
    lstWorkerWindow.mainloop()
    return

"""
```

```
        MÉTODO DE LISTAR TAREAS
    """
    def listDuties():
        global dutiesList

        listDutyWindow = Tk()
        listDutiesWindow(listDutyWindow, dutiesList)
        listDutyWindow.mainloop()
        return

    """
        MÉTODO DE MODIFICAR TRABAJADORES
    """
    def modifyWorker():
        global workersList

        returns = []

        modifyWorkerWindow = Tk()
         modifyWorkerObject = modifWorkerWindow(modifyWorkerWindow, workersList, knowledgeSet, abilitiesSet, abilitiesEva-
luation, returns)
        modifyWorkerWindow.mainloop()

        if not returns:
            returns.append(0)

        elif returns[0] is 'edit':
            tmpID = returns[1]
            for worker in workersList:
                if worker.getID() is tmpID:
                    worker.modifyName(returns[2],returns[3])
                    for know in worker.listKnowledge():
                        worker.removeKnowledge(know)
                    for know in returns[4]:
                        if len(know) > 2:
                            worker.addKnowledge(know)
                    for ab in returns[5]:
                        worker.updateAbilities(ab[0], ab[1])

        elif returns[0] is 'remove':
            tmpID = returns[1]
            for worker in workersList:
                if worker.getID() is tmpID:
                    workersList.remove(worker)

        modifyWorkerWindow.destroy()
        del modifyWorkerObject
        return

    """
        MÉTODO DE MODIFICAR TAREAS
    """
    def modifyDuty():
        global dutiesList

        returns = []

        modifyDutyWindow = Tk()
        modifyDutyObject = modifDutyWindow(modifyDutyWindow, dutiesList, knowledgeSet, abilitiesSet, abilitiesLevels, returns)
        modifyDutyWindow.mainloop()

        if returns[0] is 'edit':
            tmpID = returns[1]
            for duty in dutiesList:
                if duty.getID() is tmpID:
```

```
            duty.modifyGoal(returns[2])
            duty.modifyDuration(returns[3])

            for req in duty.listRequirements():
                duty.removeRequirements(req)
            for req in returns[4]:
                if len(req) > 2:
                    duty.addRequirements(req)
            for ab in returns[5]:
                duty.updateAbilities(ab[0], ab[1])

    elif returns[0] is 'remove':
        tmpID = returns[1]
        for duty in dutiesList:
            if duty.getID() is tmpID:
                dutiesList.remove(duty)

    modifyDutyWindow.destroy()
    del modifyDutyObject
    return


"""
    MÉTODO DE ORGANIZAR TAREAS (PREVIAS Y POSTERIORES)
"""
def prevPostDuties():
    global dutiesList

    returns = []

    dutyWindow = Tk()
    prevPostDutiesObject = prevPostDutiesWindow(dutyWindow, dutiesList, returns)
    dutyWindow.mainloop()

    for tmpOrigDuty in dutiesList:
            if returns[0] == str(tmpOrigDuty.getID()) + ': ' + str(tmpOrigDuty.getGoal()):
                origDuty = tmpOrigDuty
                for prevDuty in tmpOrigDuty.listPrevDuties():
                    tmpOrigDuty.removePrevDuty(prevDuty)
                    prevDuty.removePostDuty(tmpOrigDuty)
                for postDuty in tmpOrigDuty.listPostDuties():
                    tmpOrigDuty.removePostDuty(postDuty)
                    postDuty.removePrevDuty(tmpOrigDuty)

    for data in returns[1]:
        for checkDuty in dutiesList:
            if data == str(checkDuty.getID()) + ': ' + str(checkDuty.getGoal()):
                prevDuty = checkDuty
        origDuty.addPrevDuty(prevDuty)
        prevDuty.addPostDuty(origDuty)
        if not prevDuty.checkPrevDuty(origDuty):
            origDuty.removePrevDuty(prevDuty)
            prevDuty.removePostDuty(origDuty)

    for data in returns[2]:
        for checkDuty in dutiesList:
            if data == str(checkDuty.getID()) + ': ' + str(checkDuty.getGoal()):
                postDuty = checkDuty
        origDuty.addPostDuty(postDuty)
        postDuty.addPrevDuty(origDuty)
        if not origDuty.checkPrevDuty(postDuty):
            postDuty.removePrevDuty(origDuty)
            origDuty.removePostDuty(postDuty)

    dutyWindow.destroy()
```

```
        del prevPostDutiesObject
        return

    """
        MÉTODO DE ORGANIZAR LOS TIEMPOS A LAS TAREAS DEL PROYECTO
    """
    def orgDuties():
        for origDuty in dutiesList:
            if origDuty is not startDuty and origDuty is not endDuty:
                if not origDuty.listPrevDuties():
                    origDuty.addPrevDuty(startDuty)
                    startDuty.addPostDuty(origDuty)
                if not origDuty.listPostDuties():
                    origDuty.addPostDuty(endDuty)
                    endDuty.addPrevDuty(origDuty)

        for duty in dutiesList:
            if duty is not startDuty:
                duty.modifyDates(1, 1)
            else:
                duty.modifyDates(0, 0)

        modifFlag = True
        firstIterationFlag = True
        while modifFlag == True:
            modifFlag = False

            for duty in dutiesList:
                if duty is not startDuty:
                    listedPrevs = duty.listPrevDuties()
                    for prev in listedPrevs:
                        if (prev.getDates(1) + prev.getDuration()) > duty.getDates(1):
                            duty.modifyMinDate(prev.getDates(1) + prev.getDuration())
                            if endDuty.getDates(1) < duty.getDates(1) + duty.getDuration():
                                endDuty.modifyMinDate(duty.getDates(1) + duty.getDuration())
                            modifFlag = True


        if firstIterationFlag == True:
            for duty in dutiesList:
                duty.modifyMaxDate(endDuty.getDates(1))
            firstIterationFlag = False

        startDuty.modifyMaxDate(0)

        modifFlag = True
        while modifFlag == True:
            modifFlag = False
            for duty in dutiesList:
                listedPosts = duty.listPostDuties()
                for post in listedPosts:
                    if (post.getDates(2) - duty.getDuration()) < duty.getDates(2):
                        duty.modifyMaxDate((post.getDates(2) - duty.getDuration()))
                        modifFlag = True

        for duty in dutiesList:
            duty.calculateCriticality()

        return

    """
        MÉTODO DE AÑADIR INICIO Y FINAL AL PROYECTO
    """
    def addProjectMilestones():
        global dutiesCount
    global startDuty
    global endDuty
    startDuty = Duty(dutiesCount, "Inicio", 0)
    dutiesList.append(startDuty)
    dutiesCount = dutiesCount + 1

    endDuty = Duty(dutiesCount, "Final", 0)
    dutiesList.append(endDuty)
    dutiesCount = dutiesCount + 1

    endDuty
    return

"""
        MÉTODO PARA REPARTIR LAS TAREAS ENTRE LOS TRABAJADORES AL PROYECTO
"""
def distrDuties():
    for duty in dutiesList:
        duty.removeWorker()

    for worker in workersList:
        for duty in worker.listAssignedDuties():
            worker.freeDuty(duty)

    for duty in dutiesList:
        if duty is not startDuty and duty is not endDuty:
            if duty.getCriticality() is True:
                searchWorker(duty)

    for duty in dutiesList:
        if duty is not startDuty and duty is not endDuty:
            if duty.getCriticality() is False:
                searchWorker(duty)
    return

"""
        MÉTODO PARA ESCOGER UN TRABAJADOR PARA LA TAREA
"""
def searchWorker(duty):
    for prev in duty.listPrevDuties():
        if prev.getWorker() is None:
            searchWorker(prev)
    if duty.getWorker() == 0:
        bestScore = 0
        time = -1
        assignedWorker = None
        for worker in workersList:
            if worker.checkEfficiency(duty) > bestScore:
                time = worker.hasTime(duty)
                if time is not -1:
                    bestScore = worker.checkEfficiency(duty)
                    assignedWorker = worker
                    assignedTime = time

        if assignedWorker is not None:
            assignedWorker.assignDuty(duty, assignedTime, assignedTime + duty.getDuration() - 1)
            duty.attachWorker(assignedWorker)
            duty.setBegin(assignedTime)

"""
        MÉTODO DE CREAR EL DIAGRAMA GANNT
"""
def ganntFcn():
    ganntWindow = Tk()
    ganntWindowObject = ganntClass(ganntWindow, startDuty, endDuty)
```

```
        ganntWindow.mainloop()
        del ganntWindowObject
        return

    """
            MÉTODO DE CREAR EL DIAGRAMA PERT
    """
    def pertFcn():
        pertWindow = Tk()
        pertWindowObject = pertClass(pertWindow, startDuty)
        pertWindow.mainloop()
        del pertWindowObject
        return

    """
            MÉTODO DE EVALUACION DE COMPETENCIAS
    """
    def abilitiesEvaluation():
        abilityWindow = Tk()
        abilityWindowObject = abilityClass(abilityWindow, abilitiesSet, abilitiesValues)
        abilityWindow.mainloop()
        del abilityWindowObject
        return

    def changeProyect():
        global cursorDB
        global connectionDB
        global workersCount
        global dutiesCount
        global workersList
        global dutiesList
        global startDuty
        global endDuty
        global knowledgeSet

        changeProyectWindow = Tk()

        newDB = changeDB(changeProyectWindow)
        changeProyectWindow.destroy()
        database = newDB

        if cursorDB:
            cursorDB.close()
        if connectionDB:
            connectionDB.close()


        connectionDB = sqlite3.connect( newDB + ".db")
        cursorDB = connectionDB.cursor()
        newDataBase = checkDBTables(cursorDB)

        if not newDataBase:
            workersList = []
            workersCount = 0
            dutiesList = []
            dutiesCount = 0
            knowledgeSet = set()

        workersCount = loadDbWorkers(Worker, workersList, workersCount, cursorDB)
        dutiesCount = loadDbDuties(Duty, dutiesList, dutiesCount, cursorDB)

        if not newDataBase:
            startDuty = dutiesList[0]
            endDuty = dutiesList[1]
    for worker in workersList:
        worker.makeAssign(dutiesList)
        for know in worker.listKnowledge():
            knowledgeSet.add(know)
    for duty in dutiesList:
        duty.makePrevAssign(dutiesList)
        duty.makePostAssign(dutiesList)
        duty.makeAssign(workersList)
        for req in duty.listRequirements():
            knowledgeSet.add(req)


    fillMainWindow(mainWindow, addWorker, addDuty, listWorkers, listDuties, modifyWorker, modifyDuty,
                   prevPostDuties, orgDuties, distrDuties, ganntFcn, pertFcn,
                   database, changeProyect, saveProyect)

def saveProyect():

    storeWorkers(workersList, cursorDB);
    storeDuties(dutiesList, cursorDB);
    connectionDB.commit()

def loadAbilities():
   global abilitiesSet
   global abilitiesValues
   abConnectionDB = sqlite3.connect( "Abilities.db")
   abCursorDB = abConnectionDB.cursor()
   abilitiesValues = importAbilities(abCursorDB)
   abilitiesList = []
   for ab in abilitiesValues:
       abilitiesList.append(ab["ID"])
   abilitiesSet = set(abilitiesList)


if __name__ == '__main__':
    main()
```

1.2 Modelo (*Library.py*)

```
abilitiesSet = {'Innovación', 'Flexibilidad', 'Comunicación', 'Responsabilidad'}
abilitiesLevels = ["Ninguna", "Baja", "Media", "Alta"]
knowledgeSet = {"Mecánica", "Electrónica", "Informática", "Organización Industrial"}

"""      +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
         ++++++                                   ++++++
         ++++++             CLASE TRABAJADOR                ++++++
         ++++++                                   ++++++
         +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
"""


class Worker:
    """
        Clase que representa a un trabajador
        cada Trabajador cuenta con:
            id: identificador único que le asigna la aplicación en el constructor
            surname: Nombre del trabajador. Cadena vacía por defecto: ''
            lastName: Apellido del trabajador. Cadena vacía por defecto: ''
            knowledge: Set de conocimientos y abilidades del trabajador
            abilities: Diccionario de aptitudes (Keys) junto con su valoración (Valor)
            assignedDuties: Set de tareas asignadas al trabajador: [tarea, inicio, fin]
            busyDays: Vector/Lista que representa los dias ocupados
                Los días pares representan el comienzo de una actividad
        Los días impares representan el final de una actividad
"""

"""        ++++++++++++++++++++++++++++++++++++++++
           ++++++++++++  CONSTRUCTOR  ++++++++++++
           ++++++++++++++++++++++++++++++++++++++++

Constructor la de clase Worker, permite rellenar sus datos o dejarlos vacíos
"""
def __init__(self, t_id = -1, t_surname='', t_lastName=''):

   self.id = t_id
   self.surname  = t_surname
   self.lastName = t_lastName
   self.knowledge = set()
   self.abilities = {}
   self.assignedDuties = []
   self.busyDays = []
   return

def getID(self):
   return(self.id)

def setID(self, newID):
   self.id = newID

"""        ++++++++++++++++++++++++++++++++++++++++
           +++++++++  NOMBRE Y APELLIDO  +++++++++
           ++++++++++++++++++++++++++++++++++++++++
"""

"""
Método encargado de modificar el nombre
"""
def modifyName(self, t_surname, t_lastName):

   self.surname  = t_surname
   self.lastName = t_lastName
   return

"""
Método encargado de devolver el nombre.
Acepta un parametro de entrada que determina el retorno:
      0 (por defecto): devuelve la inicial del nombre y el apellido
      1: devuelve solo el nombre
      2: devuelve solo el apellido
      3: devuelve el nombre y el apellido
"""
def getName(self, t_returnMode = 0):
   if t_returnMode is 0:
      return self.surname[0] + '. ' + self.lastName
   if t_returnMode is 1:
      return self.surname
   if t_returnMode is 2:
      return self.lastName
   if t_returnMode is 3:
      return self.surname + ' ' + self.lastName
   return "Invalid code"


"""        ++++++++++++++++++++++++++++++++++++++++
           ++++  CONOCIMIENTO Y EXPERIENCIA  +++++
           ++++++++++++++++++++++++++++++++++++++++
"""
"""
```

```
Método encargado de añadir conocimiento
"""
def addKnowledge(self, t_knowledge):
    self.knowledge.add(t_knowledge)
    return

"""
    Método encargado de devolver una lista con todos los conocimientos
"""
def listKnowledge(self):
    t_knowledgeSet = []
    for t_knowledge in self.knowledge:
        t_knowledgeSet.append(t_knowledge)
    return t_knowledgeSet

"""
    Método encargado de eliminar un conocimiento
    Devuelve un 0 si se ha borrado el campo y un 1 si no existía
"""
def removeKnowledge(self, t_knowledge):
    t_knowledgeSet = set([t_knowledge])
    if t_knowledgeSet.issubset(self.knowledge):
        self.knowledge.remove(t_knowledge)
        return 0
    else:
        return 1


"""         +++++++++++++++++++++++++++++++++++++++
            +++++++++++++ APTITUDES  ++++++++++++++
            +++++++++++++++++++++++++++++++++++++++
"""
"""
Método encargado de añadir aptitudes
Devuelve un 0 si el campo se actualiza y un 1 si es un nuevo añadido
"""
def updateAbilities(self, t_abilities, t_value):
    updateCheck = self.abilities.get(t_abilities,-1)
    self.abilities.update({t_abilities: int(t_value)})
    if updateCheck is -1:
        return 1
    else:
        return 0

"""
    Método encargado de devolver una lista con todas las aptitudes
"""
def listAbilities(self):
    t_abilitiesKeys = []
    for key in self.abilities.keys():
        t_abilitiesKeys.append(key)
    return t_abilitiesKeys

"""
    Método encargado de acceder a la valoración de cada aptitud
    Devuelve la experiencia para un conocimiento dado
"""
def getAbilities(self, t_abilities):
    return  self.abilities.get(t_abilities)

"""
    Método encargado de eliminar un conocimiento
    Devuelve un 0 si se ha borrado el campo y un 1 si no existía
"""
def removeAbilities(self, t_abilities):
    status = self.abilities.pop(t_abilities, None)
    if status is None:
        return 1
    else:
        return 0

""""        +++++++++++++++++++++++++++++++++++++++
            ++++++++ ASIGNACIÓN DE TAREAS  ++++++++
            +++++++++++++++++++++++++++++++++++++++
"""
"""
    Método encargado de fiijar una tarea entre unos días determinados
    Comprueba si la tarea se puede asignar esos días.
        Si no se puede asignar devuelve un 1
        Si la tarea "acaba" antes de "empezar" devuelve un 2
        Si todo se asigna correctamente devuelve un 0
"""
def assignDuty(self, t_duty, t_begin, t_end):
    if t_end < t_begin:
        return 2
    if self.checkFreeDays(t_begin, t_end) is True:
        self.assignedDuties.append(t_duty)
        self.busyDays.append(t_begin)
        self.busyDays.append(t_end)
        self.busyDays.sort()
        return 0
    else:
        return 1

"""
    Método que permite asignar una tarea que no sea un puntero
    Empleado para la carga desde la base de datos
"""
def preassignDuty(self, t_duty):
    self.assignedDuties.append(t_duty)


"""
    Método encargado de comprobar si un trabajador tiene libre el intervalo entre dos días
"""
def checkFreeDays(self, t_begin, t_end):
    i = 1
    if not self.busyDays:
        return True
    try:
        while(1):
            if t_end < self.busyDays[0]:
                return True
            if (t_begin > self.busyDays[-1]):
                return True
            if (t_begin > self.busyDays[i] and t_end < self.busyDays[i+1]):
                return True
            i = i + 2
    except:

        return False

"""
    Método encargado de eliminar una de las tareas asignadas al trabajador
"""
def freeDuty(self, t_duty):
    removeDutiesList = []
    for p_duty in self.assignedDuties:
        if p_duty is t_duty:
             if p_duty.getDates(0) in self.busyDays:
```

```
            self.busyDays.remove(p_duty.getDates(0))
          if (p_duty.getDates(0) + p_duty.getDuration() - 1) in self.busyDays:
            self.busyDays.remove(p_duty.getDates(0) + p_duty.getDuration() - 1)
          removeDutiesList.append(p_duty)
    for r_duty in removeDutiesList:
      self.assignedDuties.remove(r_duty)

def removeAllDuties(self):
     self.assignedDuties = []
     return

"""
   Método encargado de devolver una lista de tareas
"""
def listAssignedDuties(self):
   t_list = []
   for duty in self.assignedDuties:
     t_list.append(duty)
   return t_list

def makeAssign(self, dutiesList):
   deletingDuties = []
   includingDuties = []
   for preasigned in self.assignedDuties:
      for duty in dutiesList:
         if preasigned == str(duty.getID()) + ': ' + duty.getGoal():
            deletingDuties.append(preasigned)
            includingDuties.append(duty)

   for preasigned in deletingDuties:
      self.assignedDuties.remove(preasigned)

   for duty in includingDuties:
      self.assignedDuties.append(duty)

"""
   Método encargado de calcular la idoneidad del trabajador a una tarea
"""
def checkEfficiency(self, duty):
   score = 0
   if duty.listRequirements() is not ['']:
      for req in duty.listRequirements():
         if req not in self.knowledge:
            return -1
   for ab in duty.listAbilities():
      if ab in self.listAbilities():
         score = score + duty.getAbilities(ab) * self.getAbilities(ab)
   return score

"""
   Método encargado de encontrar cuando se puede realizar una tarea
"""
def hasTime(self, duty):
   minBegin = duty.getDates(1)
   maxBegin = duty.getDates(2)

   for prev in duty.listPrevDuties():
      minBegin = max (minBegin, prev.getDates(0) + prev.getDuration())

   for t in range(minBegin, maxBegin + 1):
      if self.checkFreeDays(t, t + duty.getDuration()-1):
         return t
   return -1

"""
```

```
        Método encargado de devolver la lista de dias ocupados
    """
    def getBusyDays(self):
        return self.busyDays

    """
        Método encargado de fijar la lista de dias ocupados
    """
    def setBusyDays(self, newBusyDays):
        self.busyDays = []
        for day in newBusyDays:
            if day != '':
                self.busyDays.append(int(day))
        self.busyDays.sort()


"""     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
        ++++++                                      ++++++
        ++++++              CLASE TAREA              ++++++
        ++++++                                      ++++++
        ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
"""

class Duty:
    """
        Clase que representa una tarea
        cada tarea cuenta con:
            id: identificador único que le asigna la aplicación en el constructor
            goal: Meta/Objetivo tras el cual se dara la tarea por cumplida
            duration: Duracion de la tarea en días
            dates: Diccionario de fechas de inicio mínimo, máximo y prevista dentro del proyecto
            requirements: set de conocimientos y abilidades/destrezas necesarias
            abilities: Diccionario de aptitudes (Keys) junto con su importancia (Valor)
            prevDuties: set que representa todas las tareas que deben concluir antes de poder empezar esta
            postDuties: set que representa todas las tareas que dependen de esta
            criticalRoute: variable booleana que permite saber si la tarea es parte de la ruta crítica
            worker: trabajador encargado de llevar a cabo esta tarea

    """

    """         ++++++++++++++++++++++++++++++++++++++++
                ++++++++++++  CONSTRUCTOR  ++++++++++++
                ++++++++++++++++++++++++++++++++++++++++

    Constructor la de clase Duty, permite rellenar sus datos o dejarlos vacíos
    """
    def __init__(self, t_id, t_goal = '', t_duration = -1):
        self.id = t_id
        self.goal = t_goal
        self.duration = t_duration

        self.requirements = set()
        self.abilities = {}


        self.prevDuties = set()
        self.postDuties = set()
        self.dates = {'minBegin': -1, 'maxBegin': -1, 'begin': -1}
        self.criticalRoute = False

        self.worker = None
        return

    """
    Método encargado de devolver el identificador de la tarea
```

```
"""
def getID(self):
    return self.id

def setID(self, newID):
    self.id = newID

"""         +++++++++++++++++++++++++++++++++++++++
            +++++++++++++ OBJETIVO ++++++++++++++
            +++++++++++++++++++++++++++++++++++++++
"""
"""
Método encargado de modificar el objetivo de la tarea
"""
def modifyGoal(self, t_goal):
    self.goal = t_goal
    return

"""
Método encargado de devolver el objetivo de la tarea
"""
def getGoal(self):
    return self.goal

"""         +++++++++++++++++++++++++++++++++++++++
            +++++++++++++ DURACION ++++++++++++++
            +++++++++++++++++++++++++++++++++++++++
"""
"""
Método encargado de modificar la duración de la tarea
Retorna un 1 si la duración es negativa
"""
def modifyDuration(self, t_duration):
    t_duration = int(t_duration)
    if t_duration < 0:
        return 1
    self.duration = t_duration
    return

"""
Método encargado de devolver la duración de la tarea
"""
def getDuration(self):
    return self.duration

"""         +++++++++++++++++++++++++++++++++++++++
            ++++++++++++++ FECHAS +++++++++++++++
            +++++++++++++++++++++++++++++++++++++++
"""
"""
Método encargado de modificar la fecha de inicio min
"""
def modifyMinDate(self, t_minDate):
    self.dates.update({'minBegin': t_minDate})
    return

"""
Método encargado de modificar la fecha de inicio max
"""
def modifyMaxDate(self, t_maxDate):
    self.dates.update({'maxBegin': t_maxDate})
    return

"""
Método encargado de modificar las fechas de inicio min y max de la tarea
```

```
Retorna un 1 si las fechas son incompatibles
"""
def modifyDates(self, t_minDate, t_maxDate):
    if t_maxDate < t_minDate:
        return 1
    self.dates.update({'minBegin': t_minDate, 'maxBegin': t_maxDate})
    return 0

"""
Método encargado de fijar la fechas de inicio real de la tarea
Retorna un 1 si la fecha de inicio es menor al inicio min.
Retorna un 2 si la fecha de inicio es menor al inicio max.
"""
def setBegin(self, t_begin):
    if t_begin < self.dates['minBegin']:
        return 1
    if t_begin > self.dates['maxBegin']:
        return 2
    self.dates.update({'begin': t_begin})
    return

"""
Método encargado de devolver las fechas de inicio de la tarea.
Emplea un codigo de entrada para determinar el retorno:
    0 -> fecha de inicio fijada
    1 -> fecha de inicio mínimo
    2 -> fecha de inicio máximo
"""
def getDates(self, t_returnCode):
    if t_returnCode is 0:
        return self.dates['begin']
    if t_returnCode is 1:
        return self.dates['minBegin']
    if t_returnCode is 2:
        return self.dates['maxBegin']

"""         ++++++++++++++++++++++++++++++++++++++++
            ++++++++++++  REQUISITOS  +++++++++++++
            ++++++++++++++++++++++++++++++++++++++++
"""
"""
    Método encargado de añadir requisitos
"""

def addRequirements(self, t_requirement):
    self.requirements.add(t_requirement)
    return


def listRequirements(self):
    """
        Método encargado de devolver una lista con todos los requisitos
    """
    t_requirementsSet = []
    for t_requirements in self.requirements:
        t_requirementsSet.append(t_requirements)
    return t_requirementsSet

def removeRequirements(self, t_requirements):
    """
        Método encargado de borrar una destreza
        Devuelve un 0 si se ha borrado el campo y un 1 si no existía
    """
    t_requirementsSet = set(t_requirements)
    if t_requirementsSet.issubset(self.requirements):
        self.requirements.remove(t_requirements)
        return 0
    else:
        return 1

""""        +++++++++++++++++++++++++++++++++++++++
        +++++++++++++ APTITUDES  ++++++++++++++
        +++++++++++++++++++++++++++++++++++++++
""""
""""
Método encargado de añadir aptitudes
Devuelve un 0 si el campo se actualiza y un 1 si es un nuevo añadido
""""
def updateAbilities(self, t_abilities, t_value):
    updateCheck = self.abilities.get(t_abilities,-1)
    t_value = self.getAbilityValue(t_value)

    self.abilities.update({t_abilities: int(t_value)})
    if updateCheck is -1:
        return 1
    else:
        return 0

""""
Método encargado de pasar de un dato cualitativo a un dato cuantitativo
""""
def getAbilityValue(self, ab):
    if ab == "Ninguna":
        return 0
    elif ab == "Baja":
        return 3
    elif ab == "Media":
        return 6
    elif ab == "Alta":
        return 10
    else:
        return ab

""""
    Método encargado de devolver una lista con todas las aptitudes
""""
def listAbilities(self):
    t_abilitiesKeys = []
    for key in self.abilities.keys():
        t_abilitiesKeys.append(key)
    return t_abilitiesKeys

""""
    Método encargado de acceder a la valoración de cada aptitud
    Devuelve la valoración para un conocimiento dado
""""
def getAbilities(self, t_abilities):
    return  self.abilities.get(t_abilities)

""""
    Método encargado de eliminar un conocimiento
    Devuelve un 0 si se ha borrado el campo y un 1 si no existía
""""
def removeAbilities(self, t_abilities):
    status = self.abilities.pop(t_abilities, None)
    if status is None:
        return 1
    else:
        return 0
```

```
"""        ++++++++++++++++++++++++++++++++++++++++
           +++++++++ TAREAS RELACIONADAS  ++++++++++
           ++++++++++++++++++++++++++++++++++++++++
"""
"""
Método encargado de añadir tareas previas
"""
def addPrevDuty(self, t_duty):
    self.prevDuties.add(t_duty)
    return

"""
Método encargado de listar las tareas previas
"""
def listPrevDuties(self):
    t_prevDuties = []
    for t_duty in self.prevDuties:
        t_prevDuties.append(t_duty)
    return t_prevDuties

"""
Método encargado de eliminar una tarea previa
Devuelve un 0 si se ha borrado el campo y un 1 si no existía
"""
def removePrevDuty(self, t_duty):
    if t_duty in self.prevDuties:
        self.prevDuties.remove(t_duty)
        return 0
    else:
        return 1

def checkPrevDuty(self, checkDuty):
    if self == checkDuty:
        return False
    for prevDuty in self.listPrevDuties():
       if not prevDuty.checkPrevDuty(checkDuty):
           return False
    return True

"""
Método encargado de realizar la asignacion definitiva de tareas previas
"""
def makePrevAssign(self, dutiesList):
    deletingDuties = []
    includingDuties = []
    for preasigned in self.listPrevDuties():
        for duty in dutiesList:
            if preasigned == str(duty.getID()) + ': ' + duty.getGoal():
                deletingDuties.append(preasigned)
                includingDuties.append(duty)

    for preasigned in deletingDuties:
        self.removePrevDuty(preasigned)

    for duty in includingDuties:
        self.addPrevDuty(duty)

"""
Método encargado de añadir tareas posteriores
"""
def addPostDuty(self, t_duty):
    self.postDuties.add(t_duty)
    return

"""
```

```
Método encargado de listar las tareas posteriores
"""
def listPostDuties(self):
    t_postDuties = []
    for t_duty in self.postDuties:
        t_postDuties.append(t_duty)
    return t_postDuties

"""
Método encargado de eliminar una tarea posterior
Devuelve un 0 si se ha borrado el campo y un 1 si no existía
"""
def removePostDuty(self, t_duty):
    if t_duty in self.postDuties:
        self.postDuties.remove(t_duty)
        return 0
    else:
        return 1

"""
Método encargado de realizar la asignacion definitiva de tareas posteriores
"""
def makePostAssign(self, dutiesList):
    deletingDuties = []
    includingDuties = []
    for preasigned in self.listPostDuties():
        for duty in dutiesList:
            if preasigned == str(duty.getID()) + ': ' + duty.getGoal():
                deletingDuties.append(preasigned)
                includingDuties.append(duty)

    for preasigned in deletingDuties:
        self.removePostDuty(preasigned)

    for duty in includingDuties:
        self.addPostDuty(duty)

"""         ++++++++++++++++++++++++++++++++++++++++
            ++++++++++++ RUTA CRÍTICA  ++++++++++++
            ++++++++++++++++++++++++++++++++++++++++
"""
"""
Método encargado de fijar la criticidad
"""
def calculateCriticality(self):
    if self.getDates(1) == self.getDates(2):
        self.criticalRoute = True
    else:
        self.criticalRoute = False
    return

"""
Método encargado de retornar la criticidad
"""
def getCriticality(self):
    return self.criticalRoute

"""
Método manual para asignar la criticidad
"""
def setCriticality(self, newCrit):
    self.criticalRoute = newCrit

"""         ++++++++++++++++++++++++++++++++++++++++
            ++++++++ TRABAJADOR ASIGNADO  +++++++++
            +++++++++++++++++++++++++++++++++++++++
    """
    """
    Método encargado de asignarle un trabajador a la tarea
    """
    def attachWorker(self, t_worker):
        self.worker = t_worker
        return

    """
    Método encargado de retornar el  trabajador asignado
    """
    def getWorker(self):
        return self.worker

    """
    Método encargado de eliminar el trabajador asignado a la tarea
    """
    def removeWorker(self):
        self.worker = 0
        return

    def makeAssign(self, workersList):
        for worker in workersList:
            if self.getWorker() == str(worker.getID()) + ': ' + worker.getName(0):
                self.attachWorker(worker)
```

1.3 Vista (*Graphics.py*)

```
"""
from tkinter import *
from tkinter import ttk, font
from tkinter.ttk import Combobox, Checkbutton
import getpass
"""
from tkinter import Label, Button, Entry, Spinbox, Radiobutton
from tkinter.ttk import Combobox
from tkinter import Canvas, LAST
from tkinter import IntVar
from math import sqrt

"""          +++++++++++++++++++++++++++++++++++++++
            +++++++++  VENTANA PRINCIPAL  +++++++++
            +++++++++++++++++++++++++++++++++++++++
"""
def fillMainWindow(thisWindow, addWorkerFcn, addDutyFcn, listWorkersFcn, listDutiesFcn, modifyWorkerFcn, modifyDuty-
Fcn,
                         prevPostFcn, orgDutiesFcn, distrDutiesFcn, ganntFcn, pertFcn, dataBase, changeProyectFcn,
saveProyectFcn):

    actualRow = 0

    thisWindow.geometry('345x335')
    thisWindow.title("Aplicación TFG")
    #thisWindow.resizable(0,0)

    welcomeUpperText = Label(thisWindow, \
                    text="TFG de Javier Martínez\nIngeniería en Org. Industrial, EUPLA, 2019" \
                    +  "\nGestión inteligente de proyectos\n ",justify='center', font = ("Cochin", 18))
    welcomeUpperText.grid(column = 0, columnspan = 2, row = actualRow)

    actualRow = actualRow + 1
    if dataBase:
        txt = "Proyecto actual: " + dataBase
    else:
        txt = "Sin proyecto abierto"
    ProyectText = Label(thisWindow, text = txt, justify = 'center', font = ("Cochin", 14))
    ProyectText.grid(column = 0, columnspan = 2, row = actualRow)

    actualRow = actualRow + 1

      saveProyectButton = Button(thisWindow, text="Guardar proyecto", command = saveProyectFcn, font = ("Cochin", 12),
fg="blue")
    saveProyectButton.grid(column = 0,  row = actualRow)

        changeProyectButton = Button(thisWindow, text="Cambiar de proyecto", command = changeProyectFcn, font =
("Cochin", 12), fg="blue")
    changeProyectButton.grid(column = 1,  row = actualRow)

    actualRow = actualRow + 1
    workerDutyText = Label(thisWindow, text="Recursos del proyecto", justify = 'center', font = ("Cochin", 16, "underline"))
    workerDutyText.grid(column = 0, columnspan = 2, row = actualRow)

    actualRow = actualRow + 1
        addWorkerButton = Button(thisWindow, text="Añadir trabajador", command = addWorkerFcn, font = ("Cochin", 12),
fg="blue")
    addWorkerButton.grid(column = 0, row = actualRow)

    addDutyButton = Button(thisWindow, text="Añadir Tarea", command = addDutyFcn, font = ("Cochin", 12), fg="blue")
    addDutyButton.grid(column = 1, row = actualRow)

    actualRow = actualRow + 1
        listWorkerButton = Button(thisWindow, text="Listar trabajadores", command = listWorkersFcn, font = ("Cochin", 12),
fg="blue")
    listWorkerButton.grid(column = 0, row = actualRow)

    listDutyButton = Button(thisWindow, text="Listar tareas", command = listDutiesFcn, font = ("Cochin", 12), fg="blue")
    listDutyButton.grid(column = 1, row = actualRow)

    actualRow = actualRow + 1
    modifWorkerButton = Button(thisWindow, text="Editar trabajadores", command = modifyWorkerFcn, font = ("Cochin", 12),
fg="blue")
    modifWorkerButton.grid(column = 0, row = actualRow)

    modifDutyButton = Button(thisWindow, text="Editar tareas", command = modifyDutyFcn, font = ("Cochin", 12), fg="blue")
    modifDutyButton.grid(column = 1, row = actualRow)

    actualRow = actualRow + 1
    projectText = Label(thisWindow, text="Organización del proyecto", justify = 'center', font = ("Cochin", 16, "underline"))
    projectText.grid(column = 0, columnspan = 2, row = actualRow)

    actualRow = actualRow + 1
    prevPostButton = Button(thisWindow, text="Relacionar tareas", command = prevPostFcn, font = ("Cochin", 12), fg="blue")
    prevPostButton.grid(column = 0, row = actualRow)

    orgDutyButton = Button(thisWindow, text="Organizar tareas", command = orgDutiesFcn, font = ("Cochin", 12), fg="blue")
    orgDutyButton.grid(column = 1, row = actualRow)

    actualRow = actualRow + 1
    distrDutyButton = Button(thisWindow, text="Distribuir tareas", command = distrDutiesFcn, font = ("Cochin", 12), fg="blue")
    distrDutyButton.grid(column = 0, columnspan = 2, row = actualRow)



    actualRow = actualRow + 1
    diagramsText = Label(thisWindow, text="Diagramas", justify = 'center', font = ("Cochin", 16, "underline"))
    diagramsText.grid(column = 0, columnspan = 2, row = actualRow)
    actualRow = actualRow + 1
    diagramGanntButton = Button(thisWindow, text="GANNT", command = ganntFcn, font = ("Cochin", 12), fg="blue")
    diagramGanntButton.grid(column = 0, row = actualRow)

    diagramGanntButton = Button(thisWindow, text="PERT", command = pertFcn, font = ("Cochin", 12), fg="blue")
    diagramGanntButton.grid(column = 1, row = actualRow)


    return

"""         +++++++++++++++++++++++++++++++++++++++++
            ++++++  VENTANA L. TRABAJADORES  ++++++
            +++++++++++++++++++++++++++++++++++++++++
"""
def listWorkersWindow(thisWindow, workersList):

    actualRow = 0
    actualColumn = 0

    thisWindow.geometry('700x150')
    thisWindow.title("Listado de trabajadores")
    #thisWindow.resizable(0,0)

    mainText = Label(thisWindow, text="Listado de Trabajadores", font = ("Cochin", 16))
    mainText.grid(column = 0, columnspan = 3, row = actualRow)

    actualRow = actualRow + 1
    surnameText = Label(thisWindow, text="Nombre", font = ("Cochin", 12, "italic"))
    surnameText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    lastnameText = Label(thisWindow, text="Apellido", font = ("Cochin", 12, "italic"))
    lastnameText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    knowledgeText = Label(thisWindow, text="Campo/s", font = ("Cochin", 12, "italic"))
    knowledgeText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    abilitiesText = Label(thisWindow, text="Competencias", font = ("Cochin", 12, "italic"))
    abilitiesText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    dutiesText = Label(thisWindow, text="Tareas asignadas", font = ("Cochin", 12, "italic"))
    dutiesText.grid(column = actualColumn, row = actualRow)

    for worker in workersList:
        actualRow = actualRow + 1
        actualColumn = 0
        workerSurnameText = Label(thisWindow, text = worker.getName(1), font = ("Cochin", 12))
        workerSurnameText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        workerLastnameText = Label(thisWindow, text = worker.getName(2), font = ("Cochin", 12))
        workerLastnameText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        t_text =''
        for t_know in worker.listKnowledge():
            t_text = t_text + t_know + ',\n'
        t_text = t_text[:-2]
        workerKnowledgeText = Label(thisWindow, text = t_text, font = ("Cochin", 12))
        workerKnowledgeText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        t_text = ''
        for ab in worker.listAbilities():
            t_text = t_text + ab + ': ' + str(worker.getAbilities(ab)) + ',\n'
        t_text = t_text[:-2]
        workerAbilityText = Label(thisWindow, text = t_text, font = ("Cochin", 10), justify = 'left')
        workerAbilityText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        t_text = ''
        for duty in  worker.listAssignedDuties():
            t_text = t_text + duty.getGoal() + ' (' + str(duty.getDates(0)) + '-' + str(duty.getDates(0) + duty.getDuration()-1) + '),\n'
        t_text = t_text[:-2]
        workerDutiesText = Label(thisWindow, text = t_text, font = ("Cochin", 10), justify = 'left')
        workerDutiesText.grid(column = actualColumn, row = actualRow)


    actualRow = actualRow + 1
    endButton = Button(thisWindow, text="Aceptar", command = thisWindow.destroy, font = ("Cochin", 12), fg="blue")
    endButton.grid(column = 2, row = actualRow)

    return


"""         ++++++++++++++++++++++++++++++++++++++++
            +++++++++  VENTANA L. TAREAS  +++++++++
            ++++++++++++++++++++++++++++++++++++++++
"""

def listDutiesWindow(thisWindow, dutiesList):

    actualRow = 0
    actualColumn = 0

    thisWindow.geometry('600x150')
    thisWindow.title("Listado de tareas")
    #thisWindow.resizable(0,0)

    mainText = Label(thisWindow, text="Listado de tareas", font = ("Cochin", 16))
    mainText.grid(column = 0, columnspan = 3, row = actualRow)

    actualRow = actualRow + 1
    goalText = Label(thisWindow, text="Objetivo", font = ("Cochin", 12, "italic"))
    goalText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    durationText = Label(thisWindow, text="Duración", font = ("Cochin", 12, "italic"))
    durationText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    requerimentsText = Label(thisWindow, text="Requisitos", font = ("Cochin", 12, "italic"))
    requerimentsText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    abilitiesText = Label(thisWindow, text="Competencias", font = ("Cochin", 12, "italic"))
    abilitiesText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    prevDutyText = Label(thisWindow, text="Tareas previas", font = ("Cochin", 12, "italic"))
    prevDutyText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    postDutyText = Label(thisWindow, text="Tareas posteriores", font = ("Cochin", 12, "italic"))
    postDutyText.grid(column = actualColumn, row = actualRow)

    actualColumn = actualColumn + 1
    dutyDatesText = Label(thisWindow, text="Fechas", font = ("Cochin", 12, "italic"))
    dutyDatesText.grid(column = actualColumn, row = actualRow)

    for duty in dutiesList:
        actualRow = actualRow + 1
        actualColumn = 0
        dutyGoalText = Label(thisWindow, text = duty.getGoal(), font = ("Cochin", 12))
        dutyGoalText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        dutyDurationText = Label(thisWindow, text = str(duty.getDuration()), font = ("Cochin", 12))
        dutyDurationText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        t_text =''
        t_list = duty.listRequirements()
        for t_req in t_list:
            t_text = t_text + t_req + ', '
        t_text = t_text[:-1]
        dutyrequisitesText = Label(thisWindow, text = t_text, font = ("Cochin", 12))
        dutyrequisitesText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        t_list = duty.listAbilities()
        t_text_ab = ''
        for ab in t_list:
            t_text_ab = t_text_ab + ab[:4] + ': ' + str(duty.getAbilities(ab)) + ', '
        dutyAbilityText = Label(thisWindow, text = t_text_ab, font = ("Cochin", 12))
        dutyAbilityText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        t_text_prev = ''
        listPrevDuties = duty.listPrevDuties()
        for tPrevDuty in listPrevDuties:
            t_text_prev = t_text_prev + str(tPrevDuty.getGoal()) + ', '
        if not t_text_prev:
            t_text_prev = '-'
        dutyPrevText = Label(thisWindow, text = t_text_prev, font = ("Cochin", 12))
        dutyPrevText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        t_text_post = ''
        listPostDuties = duty.listPostDuties()
        for tPostDuty in listPostDuties:
            t_text_post = t_text_post + str(tPostDuty.getGoal()) + ', '
        if not t_text_post:
            t_text_post = '-'
        dutyPostText = Label(thisWindow, text = t_text_post, font = ("Cochin", 12))
        dutyPostText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        if duty.getCriticality():
            t_font = ("Cochin", 12, "bold")
        else:
            t_font = ("Cochin", 12)
        if duty.getID() > 1:
            t_text_dates =  str(duty.getDates(0)) + ' : ' + str(duty.getDates(0) + duty.getDuration()) + '  (' + str(duty.getDates(1)) +
' / ' + str(duty.getDates(2)) + ') '
            dutyDatesText = Label(thisWindow, text = t_text_dates, font = t_font)
            dutyDatesText.grid(column = actualColumn, row = actualRow)



    actualRow = actualRow + 1
    endButton = Button(thisWindow, text="Aceptar", command = thisWindow.destroy, font = ("Cochin", 12), fg="blue")
```

```
        endButton.grid(column = 2, row = actualRow)

        return

    """         +++++++++++++++++++++++++++++++++++++++
                +++++++  VENTANA AÑADIR TRAB.  ++++++++
                +++++++++++++++++++++++++++++++++++++++
    """

    class WorkerWindow:
        def __init__(self, _thisWindow, _knowledgeSet, _abilitiesSet, _absEvaluation, _returnList):
            self.thisWindow = _thisWindow
            self.returnList = _returnList
            self.knowledgeSet = _knowledgeSet
            self.abilities = _abilitiesSet
            self.absEvaluation = _absEvaluation

            self.thisWindow.geometry('450x250')
            self.thisWindow.title("Ventana de trabajador")
            #self.thisWindow.resizable(0,0)

            self.fillWindow()

            return

        def fillWindow(self):

            actualRow = 0
            actualColumn = 0
             instructionsText= Label(self.thisWindow, text="Introduzca los datos del nuevo trabajador", justify='center', font = ("Co-
chin", 18))
            instructionsText.grid(column = actualColumn, columnspan=4, row=actualRow)

            actualColumn = 0
            actualRow = actualRow + 1
            getSurnameText = Label(self.thisWindow, text="Nombre: ", font = ("Cochin", 12))
            getSurnameText.grid(column = actualColumn, row = actualRow)

            actualColumn = actualColumn + 1
            self.getSurnameEntry = Entry(self.thisWindow, width = 10)
            self.getSurnameEntry.grid(column = actualColumn, row = actualRow)

            actualColumn = actualColumn + 1
            getLastnameText = Label(self.thisWindow, text="Apellido: ", font = ("Cochin", 12))
            getLastnameText.grid(column = actualColumn, row = actualRow)

            actualColumn = actualColumn + 1
            self.getLastnameEntry = Entry(self.thisWindow, width = 10)
            self.getLastnameEntry.grid(column = actualColumn, row = actualRow)

            actualRow = actualRow + 1
            actualColumn = 0
            knowledgeText = Label(self.thisWindow, text="Áreas de conocimiento: ", font = ("Cochin", 12))
            knowledgeText.grid(column = actualColumn, columnspan = 4, row = actualRow)

            self.knowledgeCombo = [0,0]
            actualRow = actualRow + 1
            actualColumn = 0
            for i in range(0, 2):
                self.knowledgeCombo[i] = Combobox(self.thisWindow)
                self.knowledgeCombo[i]['values']= list(self.knowledgeSet)
                self.knowledgeCombo[i].grid(column = actualColumn, columnspan = 2, row = actualRow)
                actualColumn = actualColumn + 2

            actualRow = actualRow + 1
        self.abilityGenText = Label(self.thisWindow, text="Competencias:", font = ("Cochin", 12))
        self.abilityGenText.grid(column = 0, columnspan = 2, row = actualRow)

        self.abilityEvaluationButton = Button(self.thisWindow, text="Evaluación de competencias", command = self.absEvalua-
tion, font = ("Cochin", 12), fg= "blue")
        self.abilityEvaluationButton.grid(column = 2, columnspan = 2, row = actualRow)

        actualRow = actualRow + 1
        actualColumn = 0
        i = 0
        self.abilityList = [[0,0]]

        for ab in self.abilities:
            if i is not 0:
                self.abilityList.append([0,0])
            self.abilityList[i][0] = Label(self.thisWindow, text = ab, font = ("Cochin", 12))
            self.abilityList[i][0].grid(column = actualColumn, row = actualRow)

            self.abilityList[i][1] = Spinbox(self.thisWindow, from_ = 0, to = 10, width = 5)
            self.abilityList[i][1].grid(column = actualColumn + 1, row = actualRow)

            if (i % 2):
                actualRow = actualRow + 1
                actualColumn = 0
            else:
                actualColumn = actualColumn + 2
            i = i + 1

        actualColumn = 1
        actualRow = actualRow + 1
        acceptWorkerButton = Button(self.thisWindow, text="Aceptar", command = self.returnData, font = ("Cochin", 12),
fg="blue")
        acceptWorkerButton.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        acceptWorkerButton = Button(self.thisWindow, text="Cancelar", command = self.thisWindow.quit, font = ("Cochin", 12),
fg="blue")
        acceptWorkerButton.grid(column = actualColumn, row = actualRow)
        return

    def returnData(self):
        t_surnameReturn = self.getSurnameEntry.get()
        t_lastnameReturn = self.getLastnameEntry.get()

        t_knowledgeReturn = []
        for know in self.knowledgeCombo:
            t_knowledgeReturn.append(know.get())

        t_abilityReturn = []
        for ab in self.abilityList:
            t_abilityReturn.append([ab[0].cget("text"), ab[1].get()])

        self.returnList.append(t_surnameReturn)
        self.returnList.append(t_lastnameReturn)
        self.returnList.append(t_knowledgeReturn)
        self.returnList.append(t_abilityReturn)

        self.thisWindow.quit()
        return

"""          +++++++++++++++++++++++++++++++++++++++
             +++++++  VENTANA AÑADIR TAREA  ++++++++
             +++++++++++++++++++++++++++++++++++++++
"""
```

```
class DutyWindow:
    def __init__(self, _thisWindow, _knowledgeSet, _abilities, _abValues, _returnList):
        self.thisWindow = _thisWindow
        self.returnList = _returnList
        self.knowledgeSet = _knowledgeSet
        self.abilities = _abilities
        self.abValues = _abValues

        self.thisWindow.geometry('600x400')
        self.thisWindow.title("Ventana de tarea")
        #self.thisWindow.resizable(0,0)

        self.fillWindow()

        return

    def fillWindow(self):

        actualRow = 0
        actualColumn = 0
        instructionsText= Label(self.thisWindow, text="Introduzca los datos de la nueva tarea", justify='center', font = ("Cochin",
18))
        instructionsText.grid(column = actualColumn, columnspan=4, row=actualRow)

        actualColumn = 0
        actualRow = actualRow + 1
        getGoalText = Label(self.thisWindow, text="Objetivo: ", font = ("Cochin", 12))
        getGoalText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        self.getGoalEntry = Entry(self.thisWindow, width = 20)
        self.getGoalEntry.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        getDurationText = Label(self.thisWindow, text="Duración: ", font = ("Cochin", 12))
        getDurationText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        self.getDurationEntry = Entry(self.thisWindow, width = 10)
        self.getDurationEntry.grid(column = actualColumn, row = actualRow)

        actualRow = actualRow + 1
        actualColumn = 0
        knowledgeText = Label(self.thisWindow, text="Requisitos: ", font = ("Cochin", 12))
        knowledgeText.grid(column = actualColumn, columnspan = 4, row = actualRow)

        self.knowledgeCombo = [0,0]
        actualRow = actualRow + 1
        actualColumn = 0
        for i in range(0, 2):
            self.knowledgeCombo[i] = Combobox(self.thisWindow)
            self.knowledgeCombo[i]['values']= list(self.knowledgeSet)
            self.knowledgeCombo[i].grid(column = actualColumn, columnspan = 2, row = actualRow)
            actualColumn = actualColumn + 2

        actualRow = actualRow + 1
        self.abilityGenText = Label(self.thisWindow, text="Competencias:", font = ("Cochin", 12))
        self.abilityGenText.grid(column = 1, columnspan = 2, row = actualRow)

        actualRow = actualRow + 1
        actualColumn = 0
        i = 0
        self.abilityList = [[0,0]]

        for ab in self.abilities:
            if i is not 0:
                self.abilityList.append([0,0])
            self.abilityList[i][0] = Label(self.thisWindow, text = ab, font = ("Cochin", 12))
            self.abilityList[i][0].grid(column = actualColumn, row = actualRow)

            self.abilityList[i][1] = Combobox(self.thisWindow)
            self.abilityList[i][1]['values']= list(self.abValues)
            self.abilityList[i][1].grid(column = actualColumn + 1, row = actualRow)

            if (i % 2):
                actualRow = actualRow + 1
                actualColumn = 0
            else:
                actualColumn = actualColumn + 2
            i = i + 1

        actualColumn = 1
        actualRow = actualRow + 1
            acceptDutyButton = Button(self.thisWindow, text="Aceptar", command = self.returnData, font = ("Cochin", 12),
fg="blue")
        acceptDutyButton.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        acceptWorkerButton = Button(self.thisWindow, text="Cancelar", command = self.thisWindow.quit, font = ("Cochin", 12),
fg="blue")
        acceptWorkerButton.grid(column = actualColumn, row = actualRow)
        return

    def returnData(self):
        t_goalReturn = self.getGoalEntry.get()
        t_durationReturn = self.getDurationEntry.get()
        try:
            t_durationReturn = int(t_durationReturn)
        except ValueError:
            t_durationReturn = 0

        t_knowledgeReturn = []
        for know in self.knowledgeCombo:
            t_knowledgeReturn.append(know.get())

        t_abilityReturn = []
        for ab in self.abilityList:
            t_abilityReturn.append([ab[0].cget("text"), ab[1].get()])

        self.returnList.append(t_goalReturn)
        self.returnList.append(t_durationReturn)
        self.returnList.append(t_knowledgeReturn)
        self.returnList.append(t_abilityReturn)

        self.thisWindow.quit()
        return

"""         ++++++++++++++++++++++++++++++++++++++++
            ++  VENTANA MODIFICAR TRABAJADORES  +++
            ++++++++++++++++++++++++++++++++++++++++
"""

class modifWorkerWindow:
    def __init__(self, _thisWindow, _workersList, _knowledgeSet, _abilities, _absEvaluation, _returnList):
        self.thisWindow = _thisWindow
        self.workersList = _workersList
        self.returnList = _returnList
        self.knowledgeSet = _knowledgeSet
        self.abilities = _abilities
        self.absEvaluation = _absEvaluation
        self.thisWindow.geometry('500x400')
        self.thisWindow.title("Ventana de modificación de trabajadores")
        #self.thisWindow.resizable(0,0)

        self.worker = 0
        self.actualRow = 0
        self.actualColumn = 0

        self.chooseWorker()

        return

    def chooseWorker(self):

        instructionsText= Label(self.thisWindow, text = "Escoja el trabajador a modificar", justify='center', font = ("Cochin", 18))
        instructionsText.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

        comboWorkers = []
        for worker in self.workersList:
            comboWorkers.append(worker.getName(3))
        self.actualRow = self.actualRow + 1
        self.workerCombo = Combobox(self.thisWindow)
        self.workerCombo['values'] = comboWorkers
        self.workerCombo.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

        self.actualRow = self.actualRow + 1

        modifyButton = Button(self.thisWindow, text="Editar", command = self.modifyWorker, font = ("Cochin", 12), fg="blue")
        modifyButton.grid(column = self.actualColumn, row = self.actualRow)

        self.actualColumn = self.actualColumn + 1
        deleteButton = Button(self.thisWindow, text="Eliminar", command = self.deleteWorker, font = ("Cochin", 12), fg="red")
        deleteButton.grid(column = self.actualColumn, row = self.actualRow)

    def modifyWorker(self):
        self.actualRow = 3
        for worker in self.workersList:
            if (worker.getName(3)) == self.workerCombo.get():
                self.worker = worker
        self.actualRow = self.actualRow + 1

        self.actualColumn = 0
        modifySurnameText = Label(self.thisWindow, text="Nombre: ", font = ("Cochin", 12))
        modifySurnameText.grid(column = self.actualColumn, row = self.actualRow)

        self.actualColumn = self.actualColumn + 1
        self.modifySurnameEntry = Entry(self.thisWindow, width = 10)
        self.modifySurnameEntry.grid(column = self.actualColumn, row = self.actualRow)
        self.modifySurnameEntry.insert(0, self.worker.getName(1))


        self.actualColumn = self.actualColumn + 1
        modifyLastnameText = Label(self.thisWindow, text="Apellido: ", font = ("Cochin", 12))
        modifyLastnameText.grid(column = self.actualColumn, row = self.actualRow)

        self.actualColumn = self.actualColumn + 1
        self.modifyLastnameEntry = Entry(self.thisWindow, width = 10)
        self.modifyLastnameEntry.grid(column = self.actualColumn, row = self.actualRow)
        self.modifyLastnameEntry.insert(0, self.worker.getName(2))

        self.actualRow = self.actualRow + 1
        self.actualColumn = 1
        knowledgeText = Label(self.thisWindow, text="Areas de conocimiento: ", font = ("Cochin", 12))
        knowledgeText.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)
```

```
self.actualColumn = 0
self.actualRow = self.actualRow + 1
self.knowledgeComboList= []
i = 0
for know in self.worker.listKnowledge():
    self.knowledgeComboList.append(Combobox(self.thisWindow))
    self.knowledgeComboList[i]['values']= list(self.knowledgeSet)
    self.knowledgeComboList[i].grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)
    self.knowledgeComboList[i].insert(0, know)

    if i % 2:
        self.actualColumn = 0
        self.actualRow = self.actualRow + 1
    else:
        self.actualColumn = self.actualColumn + 2
    i = i + 1

while i < 4:
    self.knowledgeComboList.append(Combobox(self.thisWindow))
    self.knowledgeComboList[i]['values']= list(self.knowledgeSet)
    self.knowledgeComboList[i].grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

    if i % 2:
        self.actualColumn = 0
        self.actualRow = self.actualRow + 1
    else:
         self.actualColumn = self.actualColumn + 2
    i = i + 1

self.actualRow = self.actualRow + 1
self.abilityGenText = Label(self.thisWindow, text="Competencias:", font = ("Cochin", 12))
self.abilityGenText.grid(column = 1, columnspan = 2, row = self.actualRow)

 self.abilityEvaluationButton = Button(self.thisWindow, text="Evaluación de competencias", command = self.absEvalua-
tion, font = ("Cochin", 12), fg= "blue")
self.abilityEvaluationButton.grid(column = 2, columnspan = 2, row = self.actualRow)

self.actualRow = self.actualRow + 1
self.actualColumn = 0
i = 0
self.abilityList = [[0,0]]

for ab in self.abilities:
    if i is not 0:
         self.abilityList.append([0,0])
    self.abilityList[i][0] = Label(self.thisWindow, text = ab, font = ("Cochin", 12))
    self.abilityList[i][0].grid(column = self.actualColumn, row = self.actualRow)

    self.abilityList[i][1] = Spinbox(self.thisWindow, from_ = 0, to = 10, width = 5)
    self.abilityList[i][1].grid(column = self.actualColumn + 1, row = self.actualRow)
    self.abilityList[i][1].delete(0, "end")
    self.abilityList[i][1].insert(0, self.worker.getAbilities(ab))

    if (i % 2):
        self.actualRow = self.actualRow + 1
        self.actualColumn = 0
    else:
        self.actualColumn = self.actualColumn + 2
    i = i + 1

self.actualColumn = 1
self.actualRow = self.actualRow + 1
     acceptWorkerButton = Button(self.thisWindow, text="Aceptar", command = self.returnData, font = ("Cochin", 12),
fg="blue")
```

```
            acceptWorkerButton.grid(column = self.actualColumn, row = self.actualRow)

            self.actualColumn = self.actualColumn + 1
            cancelWorkerButton = Button(self.thisWindow, text="Cancelar", command = self.thisWindow.quit, font = ("Cochin", 12),
fg="red")
            cancelWorkerButton.grid(column = self.actualColumn, row = self.actualRow)

        def deleteWorker(self):
            for worker in self.workersList:
                if (worker.getName(3)) == self.workerCombo.get():
                    self.worker = worker
            self.returnList.append('remove')
            self.returnList.append(self.worker.getID())
            self.thisWindow.quit()

            return

        def returnData(self):
            for worker in self.workersList:
                if (worker.getName(3)) == self.workerCombo.get():
                    self.worker = worker
            self.returnList.append('edit')
            self.returnList.append(self.worker.getID())

            t_knowledgeReturn = []
            for know in self.knowledgeComboList:
                t_knowledgeReturn.append(know.get())

            t_abilityReturn = []
            for ab in self.abilityList:
                t_abilityReturn.append([ab[0].cget("text"), ab[1].get()])

            self.returnList.append(self.modifySurnameEntry.get())
            self.returnList.append(self.modifyLastnameEntry.get())
            self.returnList.append(t_knowledgeReturn)
            self.returnList.append(t_abilityReturn)

            self.thisWindow.quit()
            return

    """             ++++++++++++++++++++++++++++++++++++++++
                    +++++  VENTANA MODIFICAR TAREAS  ++++++
                    ++++++++++++++++++++++++++++++++++++++++
    """

    class modifDutyWindow:
        def __init__(self, _thisWindow, _dutiesList, _knowledgeSet, _abilities, _abLevels, _returnList):
            self.thisWindow = _thisWindow
            self.dutiesList = _dutiesList
            self.returnList = _returnList
            self.knowledgeSet = _knowledgeSet
            self.abilities = _abilities
            self.abLevels = _abLevels

            self.thisWindow.geometry('500x400')
            self.thisWindow.title("Ventana de modificación de tareas")
            #self.thisWindow.resizable(0,0)

            self.duty = 0
            self.actualRow = 0
            self.actualColumn = 0

            self.chooseDuty()

            return
```

```
def chooseDuty(self):

    instructionsText= Label(self.thisWindow, text = "Escoja la tarea a modificar", justify='center', font = ("Cochin", 18))
    instructionsText.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

    comboDuties = []
    for duty in self.dutiesList:
        comboDuties.append(duty.getGoal())
    self.actualRow = self.actualRow + 1
    self.dutyCombo = Combobox(self.thisWindow)
    self.dutyCombo['values'] = comboDuties
    self.dutyCombo.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

    self.actualRow = self.actualRow + 1

    modifyButton = Button(self.thisWindow, text="Editar", command = self.modifyDuty, font = ("Cochin", 12), fg="blue")
    modifyButton.grid(column = self.actualColumn, row = self.actualRow)

    self.actualColumn = self.actualColumn + 1
    deleteButton = Button(self.thisWindow, text="Eliminar", command = self.deleteDuty, font = ("Cochin", 12), fg="red")
    deleteButton.grid(column = self.actualColumn, row = self.actualRow)

def modifyDuty(self):
    self.actualRow = 3
    for duty in self.dutiesList:
        if (duty.getGoal()) == self.dutyCombo.get():
             self.duty = duty
    self.actualRow = self.actualRow + 1

    self.actualColumn = 0
    modifyGoalText = Label(self.thisWindow, text="Objetivo: ", font = ("Cochin", 12))
    modifyGoalText.grid(column = self.actualColumn, row = self.actualRow)

    self.actualColumn = self.actualColumn + 1
    self.modifyGoalEntry = Entry(self.thisWindow, width = 10)
    self.modifyGoalEntry.grid(column = self.actualColumn, row = self.actualRow)
    self.modifyGoalEntry.insert(0, self.duty.getGoal())


    self.actualColumn = self.actualColumn + 1
    modifyDurationText = Label(self.thisWindow, text="Duración: ", font = ("Cochin", 12))
    modifyDurationText.grid(column = self.actualColumn, row = self.actualRow)

    self.actualColumn = self.actualColumn + 1
    self.modifyDurationEntry = Entry(self.thisWindow, width = 10)
    self.modifyDurationEntry.grid(column = self.actualColumn, row = self.actualRow)
    self.modifyDurationEntry.insert(0, self.duty.getDuration())

    self.actualRow = self.actualRow + 1
    self.actualColumn = 1
    requirementsText = Label(self.thisWindow, text="Requisitos:", font = ("Cochin", 12))
    requirementsText.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

    self.actualColumn = 0
    self.actualRow = self.actualRow + 1
    self.requirementsComboList= []
    i = 0
    for know in self.duty.listRequirements():
        self.requirementsComboList.append(Combobox(self.thisWindow))
        self.requirementsComboList[i]['values']= list(self.knowledgeSet)
        self.requirementsComboList[i].grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)
        self.requirementsComboList[i].insert(0, know)

        if i % 2:
            self.actualColumn = 0
            self.actualRow = self.actualRow + 1
        else:
            self.actualColumn = self.actualColumn + 2
        i = i + 1

    while i < 4:
        self.requirementsComboList.append(Combobox(self.thisWindow))
        self.requirementsComboList[i]['values']= list(self.knowledgeSet)
        self.requirementsComboList[i].grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

        if i % 2:
            self.actualColumn = 0
            self.actualRow = self.actualRow + 1
        else:
             self.actualColumn = self.actualColumn + 2
        i = i + 1

    self.actualRow = self.actualRow + 1
    self.abilityGenText = Label(self.thisWindow, text="Competencias:", font = ("Cochin", 12))
    self.abilityGenText.grid(column = 1, columnspan = 2, row = self.actualRow)

    self.actualRow = self.actualRow + 1
    self.actualColumn = 0
    i = 0
    self.abilityList = [[0,0]]

    for ab in self.abilities:
        if i is not 0:
             self.abilityList.append([0,0])
        self.abilityList[i][0] = Label(self.thisWindow, text = ab, font = ("Cochin", 12))
        self.abilityList[i][0].grid(column = self.actualColumn, row = self.actualRow)

        self.abilityList[i][1] = Combobox(self.thisWindow)
        self.abilityList[i][1]['values']= list(self.abLevels)
        self.abilityList[i][1].grid(column = self.actualColumn + 1, row = self.actualRow)

        if (i % 2):
            self.actualRow = self.actualRow + 1
            self.actualColumn = 0
        else:
            self.actualColumn = self.actualColumn + 2
        i = i + 1

    self.actualColumn = 1
    self.actualRow = self.actualRow + 1
         acceptDutyButton = Button(self.thisWindow, text="Aceptar", command = self.returnData, font = ("Cochin", 12),
fg="blue")
    acceptDutyButton.grid(column = self.actualColumn, row = self.actualRow)

    self.actualColumn = self.actualColumn + 1
       cancelDutyButton = Button(self.thisWindow, text="Cancelar", command = self.thisWindow.quit, font = ("Cochin", 12),
fg="red")
    cancelDutyButton.grid(column = self.actualColumn, row = self.actualRow)

def deleteDuty(self):
    for duty in self.dutiesList:
        if (duty.getGoal()) == self.dutyCombo.get():
             self.duty = duty
    self.returnList.append('remove')
    self.returnList.append(self.duty.getID())
    self.thisWindow.quit()

    return
    def returnData(self):
        for duty in self.dutiesList:
            if (duty.getGoal()) == self.dutyCombo.get():
                self.duty = duty
        self.returnList.append('edit')
        self.returnList.append(self.duty.getID())

        t_requirementsReturn = []
        for req in self.requirementsComboList:
            t_requirementsReturn.append(req.get())

        t_abilityReturn = []
        for ab in self.abilityList:
            t_abilityReturn.append([ab[0].cget("text"), ab[1].get()])

        self.returnList.append(self.modifyGoalEntry.get())
        self.returnList.append(self.modifyDurationEntry.get())
        self.returnList.append(t_requirementsReturn)
        self.returnList.append(t_abilityReturn)

        self.thisWindow.quit()
        return

"""            ++++++++++++++++++++++++++++++++++++++++
            +++++  VENTANA ORGANIZAR TAREAS  ++++++
            ++++++++++++++++++++++++++++++++++++++++
"""
class prevPostDutiesWindow:
    def __init__(self, _thisWindow, _dutiesList, _returnList):
        self.thisWindow = _thisWindow
        self.returnList = _returnList
        self.dutiesList = _dutiesList

        self.prevDutiesCombo = []
        self.postDutiesCombo = []

        self.dutiesIdGoals = []
        for duty in self.dutiesList:
            self.dutiesIdGoals.append(str(duty.getID()) + ': ' + str(duty.getGoal()))

        self.thisWindow.geometry('600x400')
        self.thisWindow.title("Relacion entre tareas")
        #self.thisWindow.resizable(0,0)

        self.firstFillWindow()
        return

    def firstFillWindow(self):
        self.actualRow = 0
        self.actualColumn = 0

        instructionsText= Label(self.thisWindow, text="Seleccione la tarea deseada", justify='center', font = ("Cochin", 14))
        instructionsText.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

        self.actualColumn = self.actualColumn + 2
        self.dutiesCombo = Combobox(self.thisWindow)
        self.dutiesCombo['values']= self.dutiesIdGoals
        self.dutiesCombo.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)

        self.actualColumn = 0
        self.actualRow = self.actualRow + 1
        numPrevText = Label(self.thisWindow, text="Nº de tareas previas", justify='center', font = ("Cochin", 12))
        numPrevText.grid(column = self.actualColumn, row = self.actualRow)

        self.actualColumn = self.actualColumn + 1
        self.numPrevSpin = Spinbox(self.thisWindow, from_ = 0, to = 5, width = 5)
        self.numPrevSpin.grid(column = self.actualColumn, row = self.actualRow)

        self.actualColumn = self.actualColumn + 1
        numPostText = Label(self.thisWindow, text="Nº de tareas posteriores", justify='center', font = ("Cochin", 12))
        numPostText.grid(column = self.actualColumn, row = self.actualRow)

        self.actualColumn = self.actualColumn + 1
        self.numPostSpin = Spinbox(self.thisWindow, from_ = 0, to = 5, width = 5)
        self.numPostSpin.grid(column = self.actualColumn, row = self.actualRow)

        self.actualColumn = self.actualColumn + 1
         proceedButton = Button(self.thisWindow, text="Relacionar", command = self.secondFillWindow, font = ("Cochin", 12),
fg="blue")
        proceedButton.grid(column = self.actualColumn, row = self.actualRow)

        return

    def secondFillWindow(self):
        t = 0
        while t < 10:
            for combo in self.prevDutiesCombo:
                combo.destroy()
                self.prevDutiesCombo.remove(combo)

            for combo in self.postDutiesCombo:
                combo.destroy()
                self.postDutiesCombo.remove(combo)
            t = t + 1

        duty =  self.dutiesCombo.get()

        for t_duty in self.dutiesList:
            if self.dutiesCombo.get() == str(t_duty.getID()) + ': ' + str(t_duty.getGoal()):
                duty = t_duty

        prevDutiesNumber = 0
        prevDutiesList = []
        for t_prevDuty in duty.listPrevDuties():
            prevDutiesNumber = prevDutiesNumber + 1
            prevDutiesList.append(str(t_prevDuty.getID()) + ': ' + str(t_prevDuty.getGoal()))

        postDutiesNumber = 0
        postDutiesList = []
        for t_postDuty in duty.listPostDuties():
            postDutiesNumber = postDutiesNumber + 1
            postDutiesList.append(str(t_postDuty.getID()) + ': ' + str(t_postDuty.getGoal()))



        prevNum = max( int(self.numPrevSpin.get()), prevDutiesNumber)
        postNum = max( int(self.numPostSpin.get()), postDutiesNumber)

        self.actualColumn = 0
        self.actualRow = 3
        i = 0
        while prevNum > 0:
            self.actualRow = self.actualRow + 1
            prevNum = prevNum - 1
            actCombo = Combobox(self.thisWindow)
            actCombo['values']= self.dutiesIdGoals
            actCombo.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)
            self.prevDutiesCombo.append(actCombo)
            if i < prevDutiesNumber:
                actCombo.insert(0, prevDutiesList[i])
            i = i +1

        prevRow = self.actualRow
        i = 0
        self.actualRow = 3
        self.actualColumn = self.actualColumn + 2
        while postNum > 0:
            self.actualRow = self.actualRow + 1
            postNum = postNum - 1
            actCombo = Combobox(self.thisWindow)
            actCombo['values']= self.dutiesIdGoals
            actCombo.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)
            self.postDutiesCombo.append(actCombo)
            if i < postDutiesNumber:
                actCombo.insert(0, postDutiesList[i])
            i = i +1

        self.actualRow = max(prevRow, self.actualRow)
        self.actualColumn = 1

        self.actualRow =  self.actualRow + 1
        aceptButton = Button(self.thisWindow, text = "Confirmar", command = self.returnData, font = ("Cochin", 12), fg="blue")
        aceptButton.grid(column = self.actualColumn, row = self.actualRow)
        return

    def returnData(self):
        prevRetDuties = []
        postRetDuties = []

        self.returnList.append(self.dutiesCombo.get())

        for combo in self.prevDutiesCombo:
            prevRetDuties.append(combo.get())

        for combo in self.postDutiesCombo:
            postRetDuties.append(combo.get())

        self.returnList.append(prevRetDuties)
        self.returnList.append(postRetDuties)
        self.thisWindow.quit()
        return


"""          +++++++++++++++++++++++++++++++++++++++
             +++++++  VENTANA GRAFICO GANNT  +++++++
             +++++++++++++++++++++++++++++++++++++++
"""
class ganntClass:
    def __init__(self, _thisWindow, _firstDuty, _lastDuty):

        self.firstDuty = _firstDuty
        self.lastDuty = _lastDuty
        self.thisWindow = _thisWindow

        self.minCoordX = 100
        self.minCoordY = 50
        self.coordY = self.minCoordY
        self.thick = 10

        self.drawnDuties = set()


        self.thisWindow.geometry('800x400')
        self.thisWindow.title("Diagrama GANNT")
        #self.thisWindow.resizable(0,0)
        self.canvas = Canvas(self.thisWindow, height=400, width=800, relief = "groove", bg = "light gray")
        self.drawDuty(self.firstDuty)
        self.coordY = self.coordY + 30
        self.drawDuty(self.lastDuty)
        self.drawTime(self.lastDuty.getDates(2))
        self.canvas.pack()

        return

    def drawDuty(self, duty):
        self.drawnDuties.add(duty)
        if duty.getID() is not 0 and duty.getID() is not 1:
            minStart = duty.getDates(1) + self.minCoordX
            maxStart = duty.getDates(2) + self.minCoordX
            minEnd = duty.getDates(1) + duty.getDuration() + self.minCoordX
            maxEnd = duty.getDates(2) + duty.getDuration() + self.minCoordX
            Start =  duty.getDates(0) + self.minCoordX
            End = duty.getDates(0) + duty.getDuration() + self.minCoordX

            self.canvas.create_polygon(minStart, self.coordY - self.thick + 1, maxStart, self.coordY - self.thick + 1,
                                       maxStart, self.coordY - self.thick - 1, minStart, self.coordY - self.thick - 1,
                              outline="black", fill="light green", width=1)

            self.canvas.create_polygon(minEnd, self.coordY + self.thick + 1, maxEnd, self.coordY + self.thick + 1,
                                       maxEnd, self.coordY + self.thick - 1, minEnd, self.coordY + self.thick - 1,
                              outline="black", fill="yellow", width=1)


            if duty.getWorker():
                self.canvas.create_polygon(Start, self.coordY + self.thick - 2, End, self.coordY + self.thick - 2,
                                           End, self.coordY - self.thick + 2, Start, self.coordY - self.thick + 2,
                              outline="black", fill="light blue", width=1)

                                        self.canvas.create_text(End + 2, self.coordY, anchor = "w", font = ("Purisa", 10, "bold"), text =
(duty.getWorker().getName(0)))

            self.canvas.create_text(self.minCoordX - 5, self.coordY, anchor = "e", font = ("Purisa", 8), text = duty.getGoal())
            self.canvas.pack()
        for post in duty.listPostDuties():
            if post not in self.drawnDuties and post is not self.lastDuty:
                self.coordY = self.coordY + 30
                self.drawDuty(post)
        return

    def drawTime(self, lastTime):
        X = self.minCoordX
        Y = self.minCoordY - self.thick
        while X < self.minCoordX + lastTime:
            self.canvas.create_text(X, Y, anchor = "s", font = ("Purisa", 8), text = str(X-self.minCoordX))
            #self.canvas.create_line(300, 35, 300, 200, dash=(4, 2))(
            self.canvas.pack()
            X = X + 20


"""             +++++++++++++++++++++++++++++++++++++++
                +++++++  VENTANA GRAFICO PERT  ++++++++
                +++++++++++++++++++++++++++++++++++++++
"""
class pertClass:
    def __init__(self, _thisWindow, _firstDuty):

        self.firstDuty = _firstDuty
        self.thisWindow = _thisWindow
        self.size = 15
        self.scale = 80
        self.minPosX = 50
        self.minPosY = 150

        self.statesList = []
        self.dutiesDone = set()

        self.statesMatrix = [[0]]

        self.thisWindow.geometry('600x400')
        self.thisWindow.title("Diagrama PERT")
        #self.thisWindow.resizable(0,0)

        self.generateStates(self.firstDuty)
        self.removeDuplicates()
        self.combineSameEndings()

        self.virtualCounter = 1
        self.generateVirtualDuties()

        self.generateMatrix()



        self.canvas = Canvas(self.thisWindow, height=400, width=600)

        self.drawPoints()
        self.drawLines()
        self.drawTimes()

        self.canvas.pack()


        return

    def generateStates(self, actualDuty):
        if actualDuty in self.dutiesDone:
            return
        for prev in actualDuty.listPrevDuties():
            self.generateStates(prev)

        if actualDuty.getID():
            self.statesList.append({'endingDuties': actualDuty.listPrevDuties(),
                                    'beginingDuties': [actualDuty]})
        self.dutiesDone.add(actualDuty)

        for post in actualDuty.listPostDuties():
            self.generateStates(post)

        return

    def removeDuplicates(self):
        for originalState in self.statesList:
            for checkState in self.statesList:
                if checkState is not originalState:
                    if checkState['endingDuties'] == originalState['endingDuties']:
                        if checkState['beginingDuties'] == originalState['beginingDuties']:
                            self.statesList.remove(checkState)

        return

    def combineSameEndings(self):
        for originalState in self.statesList:
            for checkState in self.statesList:
```

```
            if checkState is not originalState:
                if checkState['endingDuties'] == originalState['endingDuties']:
                    originalState['beginingDuties'] = originalState['beginingDuties'] + checkState['beginingDuties']
                    self.statesList.remove(checkState)
    return

def generateVirtualDuties(self):
    self.searchIntersections() # Buscar todas las intersecciones
    while self.intersections:
        for origIntersection in self.intersections:
            if origIntersection:
                minIntersection = True
                for checkedIntersection in self.intersections:
                    if checkedIntersection in origIntersection:
                        minIntersection = False
                if minIntersection is True: # Seleccionar la mínima intersección
                    virtualStateFlag = True
                    for checkState in self.statesList: # Ver si hace falta estado virtual
                        if checkState['endingDuties'] == origIntersection:
                            virtualStateFlag = False
                            state = checkState
                    # Modificar el estado 'original'
                    if virtualStateFlag:
                        self.statesList.append({'endingDuties': origIntersection,
                                    'beginingDuties': [self.virtualCounter]})

                    else:
                        state['beginingDuties'].append(self.virtualCounter)

                    for changingState in self.statesList: # Modificar las tareas entrantes en los estados necesarios

                        if self.isSubList(origIntersection, changingState['endingDuties']):
                            for d in origIntersection:
                                changingState['endingDuties'].remove(d)
                            changingState['endingDuties'].append(self.virtualCounter)
                    self.virtualCounter = self.virtualCounter + 1
                if origIntersection in self.intersections:
                    self.intersections.remove(origIntersection)
                self.searchIntersections()
    return


def isSubList(self, lst1, lst2):
    return set(lst1) < set(lst2)

def isInList(self, data, lst2):
    for check in lst2:
        if data == check:
            return True
    return False

def isSameList(self, lst1, lst2):
    for data in lst1:
        sameFlag = False
        for checkData in lst2:
            if checkData == data:
                sameFlag = True
        if sameFlag is False:
            return False
    for data in lst2:
        sameFlag = False
        for checkData in lst1:
            if checkData == data:
                sameFlag = True
        if sameFlag is False:
                return False
        return True


    def searchIntersections(self):
        self.intersections = []
        for originalState in self.statesList:
            for checkState in self.statesList:
                if checkState is not originalState:
                    inters = [value for value in originalState['endingDuties'] if value in checkState['endingDuties']]
                    if inters:
                        self.intersections.append(inters)
        return

    def generateMatrix(self):
        statesToMatrix = []
        column = 0
        # Generar el primer estado
        for state in self.statesList:
            statesToMatrix.append(state)
            if state['endingDuties'] == [self.firstDuty]:
               self.statesMatrix[0][0] = {**state}
               statesToMatrix.remove(state)
               lastDuties = state['beginingDuties']

        # Bucle hasta completar los estados
        while statesToMatrix != []:
            column = column + 1
            self.addColumn(self.statesMatrix)

            # Generar lista de siguientes estados
            nextStates = []
            for state in statesToMatrix:
                nextStates.append(state)

            statesToRemove = []
            for state in nextStates:
                for duty in state['endingDuties']:
                    if duty not in lastDuties and state in nextStates:
                         statesToRemove.append(state)

            for removeState in statesToRemove:
                for nextState in nextStates:
                    if removeState == nextState:
                        nextStates.remove(removeState)


            # Calcular en que fila de la matriz se colocará el estado
            for state in nextStates:
                linkedStates = []
                for duty in state['endingDuties']:
                   for prevState in self.statesList:
                       for prevDuty in prevState['beginingDuties']:
                           if prevDuty == duty:
                               linkedStates.append(prevState)

                stateRowSum = None
                n = 0
                for prevState in linkedStates:
                    if stateRowSum:
                        stateRowSum = stateRowSum + self.getPosition(self.statesMatrix, prevState)[1]
                        n = n + 1
                    else:
                        stateRowSum = self.getPosition(self.statesMatrix, prevState)[1]
                        n = 1
                stateRow = round(stateRowSum / n)

                #ahora viene cuando se asigna a la fila
                if self.statesMatrix[column][stateRow] != 0:
                    self.addRow(self.statesMatrix, stateRow)

                self.statesMatrix[column][stateRow] = state
                statesToMatrix.remove(state)

                for duty in state['endingDuties']:
                    if duty in lastDuties:
                        lastDuties.remove(duty)
                for duty in state['beginingDuties']:
                    lastDuties.append(duty)

        for firstState in self.statesMatrix[0]:
            if firstState != 0:
                firstState['beginingDuties'] = self.firstDuty.listPostDuties()
        return

    def addColumn(self, matrix):
        numOfRows =  self.getSize(matrix)[1]
        newColumn = []
        for row in range (numOfRows):
            newColumn.append(0)
        matrix.append(newColumn)
        return

    def addRow(self, matrix, row):
        numOfColumns =  self.getSize(matrix)[0]
        for column in range (numOfColumns):
            c1 = matrix[column][:row]
            c2 = matrix[column][row:]
            matrix[column] = c1 + [0] + c2

    def getSize(self, matrix):
        return[len(matrix),len(matrix[0])]

    def getPosition(self, Matrix, data):
        numOfColumns = self.getSize(self.statesMatrix)[0]
        numOfRows = self.getSize(self.statesMatrix)[1]
        for column in range (numOfColumns):
            for row in range (numOfRows):
                if data == Matrix[column][row]:
                    return [column, row]

    def drawPoints(self):
        statusNumber = 1
        numOfColumns = self.getSize(self.statesMatrix)[0]
        numOfRows = self.getSize(self.statesMatrix)[1]
        for column in range (numOfColumns):
            for row in range (numOfRows):
                if self.statesMatrix[column][row] != 0:
                     self.canvas.create_oval(self.minPosY + column * self.scale + self.size, self.minPosX + row * self.scale + self.si-
ze,
                                  self.minPosY + column * self.scale - self.size, self.minPosX + row * self.scale - self.size,
                           outline="black", fill="white", width=2)
                     self.canvas.create_text(self.minPosY + column * self.scale, self.minPosX + row * self.scale, anchor = "center",
font = ("Purisa", 12), text = 'S' + str(statusNumber))

                    statusNumber = statusNumber + 1
        return

    def drawLines(self):
        dutyNumber = 1
    leyendCounter = 10
    numOfColumns = self.getSize(self.statesMatrix)[0]
    numOfRows = self.getSize(self.statesMatrix)[1]
    for column in range (numOfColumns):
        for row in range (numOfRows):
            origStatus = self.statesMatrix[column][row]

            if origStatus != 0:
                for origDuty in origStatus['beginingDuties']:
                    for receiverStatus in self.statesList:
                        for receiverDuty in receiverStatus['endingDuties']:
                            if receiverDuty == origDuty:
                                if type(origDuty) is int:
                                    dashStyle = (2, 2)
                                    linewidth = 1

                                else:
                                    dashStyle = (255, 1)
                                    if origDuty.getCriticality():
                                      linewidth = 3
                                    else:
                                      linewidth = 1


                                receiverColumn = self.getPosition(self.statesMatrix, receiverStatus)[0]
                                receiverRow = self.getPosition(self.statesMatrix, receiverStatus)[1]
                                 self.canvas.create_line(self.minPosY + column * self.scale + self.size, self.minPosX + row * self.sca-
le,
                                                  self.minPosY + receiverColumn * self.scale- self.size, self.minPosX + receiverRow *
self.scale,
                                                arrow = LAST, dash = dashStyle, width= linewidth)


                                # Leyenda
                                if type(origDuty) is not int:
                                          hip = sqrt((column - receiverColumn)*(column - receiverColumn) + (row - receiverRow)*(row -
receiverRow))
                                    dispX = ((column - receiverColumn)/hip) * 8
                                    dispY = ((row - receiverRow)/hip) * 8
                                    self.canvas.create_text(self.minPosY + (column + receiverColumn) * 0.5 * self.scale - dispY,
                                                self.minPosX + (row + receiverRow) * 0.5 * self.scale + dispX,
                                                anchor = "center", font = ("Purisa", 10), text = 'T' + str(dutyNumber))

                                     self.canvas.create_text(10, self.minPosX + leyendCounter, anchor = "w", font = ("Purisa", 12), text
= 'T' + str(dutyNumber) + ' = ' + origDuty.getGoal())

                                    leyendCounter = leyendCounter + 12
                                    dutyNumber = dutyNumber + 1
    return



  def drawTimes(self):
    numOfColumns = self.getSize(self.statesMatrix)[0]
    numOfRows = self.getSize(self.statesMatrix)[1]
    for column in range (numOfColumns):
        for row in range (numOfRows):
            origStatus = self.statesMatrix[column][row]
            if type(origStatus) is not int:
                maxTimesList = []
                for duty in origStatus['beginingDuties']:
                    if type(duty) is not int:
                        maxTimesList.append(int(duty.getDates(2)))
                maxTime = min (maxTimesList)
                minTimesList = []
            for duty in origStatus['beginingDuties']:
                if type(duty) is not int:
                    minTimesList.append(int(duty.getDates(1)))
            minTime = max (minTimesList)



            self.canvas.create_rectangle(self.minPosY + column * self.scale + self.size * 1.3, self.minPosX + row * self.sca-
le - self.size * 2,
                                         self.minPosY + column * self.scale, self.minPosX + row * self.scale - self.size * 1.1,
                                         outline="black", fill="white", width = 0.5)

             self.canvas.create_text(self.minPosY + column * self.scale + self.size * 0.65, self.minPosX + row * self.scale -
self.size * 1.55,
                                    anchor = "center", font = ("Purisa", 12), text =  str(maxTime))

            self.canvas.create_rectangle(self.minPosY + column * self.scale, self.minPosX + row * self.scale - self.size * 2,
                                        self.minPosY + column * self.scale - self.size * 1.3, self.minPosX + row * self.scale - self.size
* 1.1,
                                        outline="black", fill="white", width = 0.5)

              self.canvas.create_text(self.minPosY + column * self.scale - self.size * 0.65, self.minPosX + row * self.scale -
self.size * 1.55,
                                    anchor = "center", font = ("Purisa", 12), text =  str(minTime))


class abilityClass:
    def __init__(self, _thisWindow, _abilitiesList, _AbEvalData):
        self.thisWindow = _thisWindow
        self.abilitiesList = _abilitiesList
        self.AbEvalData = _AbEvalData

        self.thisWindow.geometry('600x400')
        self.thisWindow.title("Ventana de evaluación de competencias")
        #self.thisWindow.resizable(0,0)

        self.actualRow = 0
        self.actualColumn = 0

        self.choooseAbility()

        return

    def choooseAbility(self):

        instructionsText= Label(self.thisWindow, text="Seleccione una competencia", justify='center', font = ("Cochin", 14))
        instructionsText.grid(column = self.actualColumn, columnspan = 2, row = self.actualRow)
        self.actualColumn = self.actualColumn + 2

        self.abCombo = Combobox(self.thisWindow)
        self.abCombo['values'] = list(self.abilitiesList)
        self.abCombo.grid(column = self.actualColumn, row = self.actualRow)

        self.actualRow = self.actualRow + 1
        selectButton = Button(self.thisWindow, text="Proceder", command = self.fillWindow, font = ("Cochin", 12), fg="blue")
        selectButton.grid(column = 1, columnspan  = 2, row = self.actualRow)

    def fillWindow(self):
        self.actualColumn = 0
        self.actualRow = 2
        selectedAb = 0
        for abDict in self.AbEvalData:
            if abDict["ID"] == self.abCombo.get():
                selectedAb = abDict
        if selectedAb == 0:
                errorLabel = Label(self.thisWindow, text="Competencia no encontrada", justify='center', font = ("Cochin", 14))
                errorLabel.grid(column = self.actualColumn, columnspan = 4, row = self.actualRow)
                self.actualRow = self.actualRow - 1
            else:
                question = 0
                self.RadiobuttonMatrix = []
                self.RadiobuttonVariables = []
                for questionText in selectedAb["Questions"]:

                    self.actualColumn = 0
                    self.actualRow = self.actualRow + 1

                    questionLabel = Label(self.thisWindow, text = '\n'+ questionText, justify='center', font = ("Cochin", 14))
                    questionLabel.grid(column = self.actualColumn, columnspan = 4, row = self.actualRow)

                    self.actualRow = self.actualRow + 1

                    RadiobuttonList = []
                    option = 0
                    self.RadiobuttonVariables.append(IntVar(self.thisWindow, 0))
                    for answer in selectedAb["Answers"][question]:
                        RadiobuttonList.append(Radiobutton(self.thisWindow, text = answer, variable = self.RadiobuttonVariables[ques-
tion], value = selectedAb["Values"][question][option]))
                        RadiobuttonList[option].grid(column = self.actualColumn, row = self.actualRow)

                        option = option + 1
                        self.actualColumn = self.actualColumn + 1
                    self.RadiobuttonMatrix.append(RadiobuttonList)
                    question = question + 1

                self.actualRow = self.actualRow + 1
                   evaluateButton = Button(self.thisWindow, text="Calcular", command = self.evaluateAbility, font = ("Cochin", 12),
fg="blue")
                evaluateButton.grid(column = 1, columnspan  = 2, row = self.actualRow)

        def evaluateAbility(self):
            self.actualRow = self.actualRow + 1
            sumatory = 0
            cunatity = 0
            for data in self.RadiobuttonVariables:
                sumatory = sumatory + data.get()
                cunatity = cunatity + 1

              evaluateLabel = Label(self.thisWindow, text = "La puntuación es: " + str(sumatory/cunatity), justify='center', font =
("Cochin", 20, "bold"))
            evaluateLabel.grid(column = 1, columnspan = 2, row = self.actualRow)
            self.actualRow = self.actualRow - 1

    def changeDB(thisWindow):
        thisWindow.geometry('300x50')
        thisWindow.title("Ventana de selección de proyecto")
        #self.thisWindow.resizable(0,0)

        actualRow = 0
        actualColumn = 0

        ProyectText = Label(thisWindow, text = "Proyecto: ", justify = 'center', font = ("Cochin", 14))
        ProyectText.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        ProyectEntry = Entry(thisWindow, width = 10)
        ProyectEntry.grid(column = actualColumn, row = actualRow)

        actualColumn = actualColumn + 1
        acceptButton = Button(thisWindow, text="Aceptar", command = thisWindow.quit, font = ("Cochin", 12), fg="blue")
    acceptButton.grid(column = actualColumn, columnspan  = 2, row = actualRow)

    thisWindow.mainloop()

    return ProyectEntry.get()
```

1.4 Interfaz de la base de datos (*DbInterface.py*)

```
def checkDBTables(cursorDB):
    new = False
    cursorDB.execute(''' SELECT count(name) FROM sqlite_master WHERE type='table' AND name='workersTable' ''')
    if cursorDB.fetchone()[0] == 0:
        new = True
        cursorDB.execute( """ CREATE TABLE workersTable (worker_id INTEGER PRIMARY KEY,
                                        surname VARCHAR,
                                        lastname VARCHAR,
                                        knowledge BLOB,
                                        abilities BLOB,
                                        assignedDuties BLOB,
                                        busyDays BLOB);""")

    cursorDB.execute(''' SELECT count(name) FROM sqlite_master WHERE type='table' AND name='dutiesTable' ''')
    if cursorDB.fetchone()[0] == 0:
        new = True
        cursorDB.execute( """ CREATE TABLE dutiesTable (duty_id INTEGER PRIMARY KEY,
                                        goal VARCHAR,
                                        duration INTEGER,
                                        dates BLOB,
                                        requirements BLOB,
                                        abilities BLOB,
                                        prevDuties BLOB,
                                        postDuties BLOB,
                                        criticalRoute BIT,
                                        worker BLOB);""")
    return new

def workerToDB(worker):
    cmdIntro = 'INSERT INTO workersTable VALUES ('
    cmdValues1 = str(worker.getID()) + ', "' + worker.getName(1) + '", "' + worker.getName(2)

    cmdValues2 = '", "'
    for know in worker.listKnowledge():
        cmdValues2 = cmdValues2 + know + ', '
    if worker.listKnowledge():
        cmdValues2 = cmdValues2 [:-2]


    cmdValues3 = '", "'
    for ab in worker.listAbilities():
        cmdValues3 = cmdValues3 + ab + ': ' + str(worker.getAbilities(ab)) + ', '
    if worker.listAbilities():
        cmdValues3 = cmdValues3 [:-2]


    cmdValues4 = '", "'
    for duty in worker.listAssignedDuties():
        cmdValues4 = cmdValues4 + str(duty.getID()) + ': ' + duty.getGoal() + ', '
    if worker.listAssignedDuties():
        cmdValues4 = cmdValues4 [:-2]


    cmdValues5 = '", "'
```

```
        for date in worker.getBusyDays():
            cmdValues5 = cmdValues5 + str(date) + ', '
        if worker.getBusyDays():
            cmdValues5 = cmdValues5[:-2]
        cmdValues5 = cmdValues5 + "'"

        return cmdIntro + cmdValues1 + cmdValues2 + cmdValues3 + cmdValues4 + cmdValues5 + ');'

    def dbToWorker(worker, data):
        worker.setID(data[0])
        worker.modifyName(data[1], data[2])

        knowledgeList = data[3].split(', ')
        for know in knowledgeList:
            if know != '':
                worker.addKnowledge(know)

        AbilitiesList = data[4].split(', ')
        for ab in AbilitiesList:
            if ab != '':
                abData = ab.split(': ')
                worker.updateAbilities(abData[0], abData[1])

        DutiesList = data[5].split(', ')
        for duty in DutiesList:
            if duty != '':
                worker.preassignDuty(duty)

        worker.setBusyDays(data[6].split(', '))


    def dutyToDB(duty):
        cmdIntro = 'INSERT INTO dutiesTable VALUES ('
        cmdValues1 = str(duty.getID()) + ', "' + duty.getGoal() + '", "' + str(duty.getDuration())

        cmdValues2 = '", "' + str(duty.getDates(0)) + ', ' + str(duty.getDates(1)) + ', ' + str(duty.getDates(2))

        cmdValues3 = '", "'
        for req in duty.listRequirements():
            cmdValues3 = cmdValues3 + req + ', '
        if duty.listRequirements():
            cmdValues3 = cmdValues3 [:-2]


        cmdValues4 = '", "'
        for ab in duty.listAbilities():
            cmdValues4 = cmdValues4 + ab + ': ' + str(duty.getAbilities(ab)) + ', '
        if duty.listAbilities():
            cmdValues4 = cmdValues4 [:-2]


        cmdValues5 = '", "'
        for prevDuty in duty.listPrevDuties():
            cmdValues5 = cmdValues5 + str(prevDuty.getID()) + ': ' + prevDuty.getGoal() + ', '
        if duty.listPrevDuties():
            cmdValues5 = cmdValues5 [:-2]

        cmdValues6 = '", "'
        for postDuty in duty.listPostDuties():
            cmdValues6 = cmdValues6 + str(postDuty.getID()) + ': ' + postDuty.getGoal() + ', '
        if duty.listPostDuties():
            cmdValues6 = cmdValues6 [:-2]


        cmdValues7 = '", "' + str(duty.getCriticality()) + '"'
        if duty.getWorker():
            worker = duty.getWorker()
            cmdValues8 = ', "' + str(worker.getID()) + ': ' + worker.getName(0) + '"'
        else:
            cmdValues8 = ',""'

        return cmdIntro + cmdValues1 + cmdValues2 + cmdValues3 + cmdValues4 + cmdValues5 + cmdValues6 + cmdValues7 +
cmdValues8 + ');'

    def dbToDuty(duty, data):
        duty.setID(data[0])
        duty.modifyGoal(data[1])
        if data[2] != '':
            duty.modifyDuration(data[2])
        else:
            duty.modifyDuration(-1)

        datesList = data[3].split(', ')
        duty.modifyDates(int(datesList[1]), int(datesList[2]))
        duty.setBegin(int(datesList[0]))

        requirementsList = data[4].split(', ')
        for req in requirementsList:
            if req != '':
                duty.addRequirements(req)

        AbilitiesList = data[5].split(', ')
        for ab in AbilitiesList:
            abData = ab.split(': ')
            if ab != '':
                duty.updateAbilities(abData[0], abData[1])

        prevList = data[6].split(', ')
        for prevDuty in prevList:
            if prevDuty != '':
                duty.addPrevDuty(prevDuty)

        postList = data[7].split(', ')
        for postDuty in postList:
            if postDuty != '':
                duty.addPostDuty(postDuty)

        if data[8]  == "True":
            duty.setCriticality(True)
        else:
             duty.setCriticality(False)

        if data[9] != '':
            duty.attachWorker(data[9])


    def loadDbWorkers(Worker, workers, counter, cursor):
        cursor.execute("SELECT * FROM workersTable")
        workersDB = cursor.fetchall()
        for row in workersDB:
            workers.append(Worker())
            dbToWorker(workers[-1], row)
            counter = counter + 1
        return counter


    def loadDbDuties(Duty, duties, counter, cursor):
        cursor.execute("SELECT * FROM dutiesTable")
        dutiesDB = cursor.fetchall()
    for row in dutiesDB:
        duties.append(Duty(-1))
        dbToDuty(duties[-1], row)
        counter = counter + 1
    return counter

def storeWorkers(workersList, cursor):

    cursor.execute('DELETE FROM workersTable')
    for worker in workersList:
        cursor.execute(workerToDB(worker))

def storeDuties(dutiesList, cursor):
    cursor.execute("DELETE FROM dutiesTable")
    for duty in dutiesList:
        cursor.execute(dutyToDB(duty))

def importAbilities(cursor):
    absGlobal = []
    cursor.execute('SELECT name from sqlite_master where type= "table"')
    ans = cursor.fetchall()
    for ab in ans:
       cursor.execute("SELECT * FROM " + ab[0])
       tmpID = ab[0]
       data = cursor.fetchall()
       tmpQuestion = []
       tmpAnswers = []
       tmpValues = []
       for row in data:
           tmpQuestion.append(row[0])

           answersList = row[1].split(', ')
           tmpAnswers.append(answersList)

           valuesList = row[2].split(', ')
           tmpValues.append(valuesList)
       tmpDict = { "ID": tmpID, "Questions" : tmpQuestion, "Answers" : tmpAnswers, "Values" : tmpValues }
       absGlobal.append(tmpDict)
    return absGlobal
```

1.5 Base de datos de competencias (*AbilitiesEvaluation.py*)

```
InnID = "Innovación"
InnQuestions = ["Pregunta 1",
                "Pregunta 2",
                "Pregunta 3",
                "Pregunta 4",]
InnAnswers = [["A", "B", "C", "D"],
              ["A", "B", "C", "D"],
              ["A", "B", "C", "D"],
              ["A", "B", "C", "D"]]

InnAnswersValues = [[0, 3, 6 , 10],
                    [0, 3, 6 , 10],
                    [0, 3, 6 , 10],
                    [0, 3, 6 , 10]]

InnDict = { "ID": InnID,
            "Questions" : InnQuestions,
            "Answers" : InnAnswers,
            "Values" : InnAnswersValues }

#--------------------------------------------------------------------------------
```

```
#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------

FlexID = "Flexibilidad"
FlexQuestions = ["Pregunta 1",
              "Pregunta 2",
              "Pregunta 3",
              "Pregunta 4",]
FlexAnswers = [["A", "B", "C", "D"],
            ["A", "B", "C", "D"],
            ["A", "B", "C", "D"],
            ["A", "B", "C", "D"]]

FlexAnswersValues = [[0, 3, 6 , 10],
                  [0, 3, 6 , 10],
                  [0, 3, 6 , 10],
                  [0, 3, 6 , 10]]

FlexDict = { "ID": FlexID,
          "Questions" : FlexQuestions,
          "Answers" : FlexAnswers,
          "Values" : FlexAnswersValues }

#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------

ComID = "Liderazgo"
ComQuestions = ["Empuja a la consecución de buenos resultados",
              "Transmite sentido de urgencia cuando es necesario",
              "Dirige y trabaja para resolver problemas",
              "Crea ambiente para que se disfrute trabajando",
              "Consigue facilmente la confianza y el respeto de los demás"]
ComAnswers = [["Insuficiente", "Normal", "Satisfactorio", "Bueno", "Excelente"],
            ["Insuficiente", "Normal", "Satisfactorio", "Bueno", "Excelente"],
            ["Insuficiente", "Normal", "Satisfactorio", "Bueno", "Excelente"],
            ["Insuficiente", "Normal", "Satisfactorio", "Bueno", "Excelente"],
            ["Insuficiente", "Normal", "Satisfactorio", "Bueno", "Excelente"]]

ComAnswersValues = [[0, 2, 4 , 7, 10],
                 [0, 2, 4 , 7, 10],
                 [0, 2, 4 , 7, 10],
                 [0, 2, 4 , 7, 10],
                 [0, 2, 4 , 7, 10]]

ComDict = { "ID": ComID,
          "Questions" : ComQuestions,
          "Answers" : ComAnswers,
          "Values" : ComAnswersValues }

#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------
#--------------------------------------------------------------------------------

RespID = "Responsabilidad"
RespQuestions = ["Pregunta 1",
              "Pregunta 2",
              "Pregunta 3",
              "Pregunta 4",]
RespAnswers = [["A", "B", "C", "D"],
            ["A", "B", "C", "D"],
            ["A", "B", "C", "D"],
            ["A", "B", "C", "D"]]

RespAnswersValues = [[0, 3, 6 , 10],
                [0, 3, 6 , 10],
                [0, 3, 6 , 10],
                [0, 3, 6 , 10]]

RespDict = { "ID": RespID,
        "Questions" : RespQuestions,
        "Answers" : RespAnswers,
        "Values" : RespAnswersValues }

#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
AbEvalData = [InnDict, FlexDict, ComDict, RespDict]

def createAbsDB():
    import sqlite3

    abConnectionDB = sqlite3.connect( "Abilities.db")
    abCursorDB = abConnectionDB.cursor()

    for ab in AbEvalData:
        abCursorDB.execute("SELECT count(name) FROM sqlite_master WHERE type='table' AND name='" + ab["ID"]+"'")
        if abCursorDB.fetchone()[0]:
            abCursorDB.execute('DELETE FROM ' + ab["ID"])
        else:
            cmd = "CREATE TABLE " + ab["ID"] + " (Question BLOB, Answers BLOB, Valorations BLOB);"
            abCursorDB.execute(cmd)

        questionNumber = 0
        for question in ab["Questions"]:
            cmd = "INSERT INTO " + ab["ID"] + " VALUES ('" + question + "', '"
            for answer in ab["Answers"][questionNumber]:
                cmd = cmd + answer + ", "
            cmd = cmd[:-2] + "','"
            for value in ab["Values"][questionNumber]:
                cmd = cmd + str(value) + ", "
            cmd = cmd[:-2] + "');"
            abCursorDB.execute(cmd)
            questionNumber = questionNumber + 1
    abConnectionDB.commit()

createAbsDB()
```

Relación de documentos

(_) Memoria ....67 .....páginas

(X) Anexos ....50 .....páginas

La Almunia, a 26 de Noviembre de 2019

Firmado: Javier Martínez Lahoz