System Maintenance

1. Environment

1.1 Software

This is a complete list of the software I used when creating my program:

- Python 3 (the programming language)
- IDLE (to write Python scripts)
- PyQt 4 (to produce user interface elements)
- SQLite 3 (for databases)
- CX_Freeze (for compling my program)
- Microsoft Paint (for creating graphics)
- Adobe Fireworks CS5 (for adding transparency to graphics)

1.2 Usage Explanation

Python 3:

- The language I'm most familiar and practiced in
- Python is open source, meaning it's free to use however I like so there no restrictions, including for commercial use.
- Compatible with all systems, so should my client change system at any point, the program can still be used.
- There is a vast number of pre-existing modules in library, meaning I was able to use a lot of modules from that
- It's an object orientated programming language

IDLE:

- Designed especially for python scripting
- Easy to use as there is no complicated interfere or obscure features
- Colour coded text and automatic indentation to make programming easier
- Has a debug feature making testing easier

PyQ 4t:

- Designed specifically for implementing GUI in python

SQLite 3

- More simplified version of MySQL, making it easier to use as there are fewer features

- Does not require a log in or password. This is fine as my client asked for all areas of the program to be accessible by anyone (see section 1.4 in Project Analysis). It also makes the building and testing of the system easier and faster as you do not have to log in each time
- Compatible with Python 3

CX_Freeze:

- Used for compiling python files
- Compatible with Python 3, unlike most python compilers
- Compiles all modules used with the program, so nothing needed to install and use the resulting program
- Includes the option to include other files when creating the compiled program.

Microsoft Paint

- Easy to create the simply graphics I wanted
- Do not have to export images
- Good for creating low detailed, small graphics

Adobe Fireworks CS5

- Wand tool allows for simple selection and deletion of areas, making image transparency easy to create
- Image can be saved straight over original, unlike with Photoshop which requires exporting the image first.

1.3 Features Used

Python 3:

- This is the language I used to write my program so naturally it is the main basis for it. Specifically I used the built in random module for generating some random numbers in my code in a specified range.

IDLE:

- I used this to write my python script in and run it
- I used the colour coded text and automatic indentation to make programming easier

PyQt 4:

- I used this to create the GUI for my program. I utilized the QtCore and QtGui modules. Specifically, I used the module for creating planes (with QRectF) and for specifying the size of objects, ect. (with QSize).
- I used the QtGui modules for creating all my widgets and windows, as well as the graphical displays. The menu bar was also created using this module (QMenuBar).

SQLite 3:

- I used the SQLite syntax for creating, reading and writing to and from my database.

CX_Freeze:

- I used this to compile all my program files into aWindows installer file.
- I made use of the ability to compile other files, specifically image files, along with the program files.

Microsoft Paint:

- I used this to create the graphics for my program that I read from image files.

Adobe Fireworks CS5:

- I used this simply for the wand tool to make my images I made in Paint transparent.

2. System Overview

2.1 The Graphic Display

The graphic display is made up of a visual representation of the fish and plants present in the pond, a text box containing the total number and type of the plants and fish currently in the pond, a text box containing the status of the pump and filter, and finally a visual representation of the pond's water level and temperature.

All the data required for the graphic displayed is read from the database after the user has either created a new pond or selected the pond they wish to open from the 'open pond' window. The data required for the display is as followed: pond water level, pond temperature, pump status of pump matching pump ID in pond, filter status of filter matching filter ID in pond, a list of fish with pond ID matching the ID of the pond, a list of plants with pond ID matching the ID of the pond.

2.2 Manipulating the Pond Object

In order for the program to be useful, the user must be able to manipulate the pond object. In order to create a new pond, the user must select the "New Pond" option in the file menu. From there, the information is filled out and the pond is saved to the database as well as opening in the program (visible in the graphic display).

An existing pond may be opened by clicking the "open pond" option. From here, the user must select a pond from the list by double clicking on it, and the pond's information will be read into the program from the database.

An existing pond's settings may be changed by selecting the "Edit pond settings" option from the manage pond menu. After the information has been inputted by the user in the same way as making a new pond, it's saved to the pond attributes with a series of setter functions. The database is then updated with an update function, writing over the record.

2.3 Adding and removing fish and plants

Fish and plants can be added and removed from the pond at any time. There are 4 different types of plants – rooted floating plant, true floating plant, emergent plant and submersed plant – and 6 different types of fish – goldfish, tench, orfe, koi, rudd and sharks. All fish types have different growth rates, food requirements and starting sizes. All plant types have different growth rates and light needs.

To add a fish, the user must select "Add fish" from the manage pond life menu in the manage pond menu, then select the type of fish they wish to add from the drop down menu and enter a name. The fish will be stored in the database and a graphical representation is added to the pond graphic.

To remove a fish, the user must select "remove fish" from the manage pond life menu in the manage pond menu, then select the fish they wish to remove by double clicking on an item from the displayed list.

To add a plant to the pond, the user must select "Add plant" from the manage pond life menu in the manage pond menu, then select the type of plant they wish to add from the drop down menu. The plant will be stored in the database and a graphical representation is added to the pond graphic.

To remove a plant, the user has to select "remove plant" from the manage pond life menu in the manage pond menu, then double click on the plant they wish to from the list provided.

2.4 Feeding fish

Fish need to be fed in order for them to live and grow. There are two ways to feed fish in the system. The first way is manually. Manual feeding asks the user for an amount of food they'd like to feed the fish with. It then feeds all the fish in the pond using that amount of food. If there's not enough to feed all the fish, some fish will remain hungry. Feeding the fish manually increments the day by 1.

The second way to feed fish in the pond is by automatically feed them. Automatically feeding calculates the exact amount of food the fish need and feeds them for 7 days.

3. Code Structure

3.1 Function – openWindow()

```
def openWindow():
#when pond is opened
def onOpen():
    self.window.close() #hides window
    self.selected=self.window.tree.currentItem()
    pondID=self.selected.data(0,0)
    depth=self.selected.data(1,0)
    length=self.selected.data(2,0)
    width=self.selected.data(3,0)
    temp=self.selected.data(4,0)
    level=self.selected.data(5,0)
    pump=self.selected.data(6,0)
    Filter=self.selected.data(7,0)
    days=self.selected.data(8,0)

self.mainPond=pond.Pond(pondID=pondID,depth=depth,width=width,temp=temp,wat
erLevel=level,pump=pump,filter=Filter,days=days, length=length)
    self.createUI()

def tree():
    self.window.tree.setHeaderItem(self.window.header) #setting header
    results=pondSimDatabase.getPonds(db,cursor)
    for count in range(0,len(results)):
        self.window.treeList=QTreeWidgetItem()#list items
        self.window.treeList.setText(0,str(results[count][0]))
        self.window.treeList.setText(1,str(results[count][1]))
        self.window.treeList.setText(2,str(results[count][2]))
        self.window.treeList.setText(3,str(results[count][3]))
        self.window.treeList.setText(4,str(results[count][4]))
        self.window.treeList.setText(5,str(results[count][5]))
        self.window.treeList.setText(6,str(results[count][6]))
        self.window.treeList.setText(7,str(results[count][7]))
        self.window.treeList.setText(8,str(results[count][8]))
        self.window.tree.addTopLevelItem(self.window.treeList)

self.window=QWidget()
self.window.setWindowTitle("Pond Simulator 3000 - Open Pond")
self.window.setFixedWidth(600)
self.window.setFixedHeight(400)
self.window.show()
self.window.layout = QVBoxLayout()
self.window.setLayout(self.window.layout)

self.window.tree=QTreeWidget()
self.window.tree.setColumnCount(9)
self.window.tree.setColumnWidth(0,80)

self.window.header=QTreeWidgetItem() #headers
self.window.header.setText(0,'PondID')
self.window.header.setText(1,'Pond Depth')
self.window.header.setText(2,'Pond Length')
self.window.header.setText(3,'Pond Width')
self.window.header.setText(4,'Water Temp.')
self.window.header.setText(5,'Water Level')
self.window.header.setText(6,'PumpID')
self.window.header.setText(7,'FilterID')
self.window.header.setText(8,'Days')
tree()

self.window.button=QPushButton("Cancel") #making button
self.window.button.setFixedWidth(100) #changing button size
self.window.button.layout=QHBoxLayout() #creating layout for button
self.window.buttonWidget=QWidget() #creating blank widget
self.window.blank=QWidget()
self.window.buttonWidget.setLayout(self.window.button.layout) #setting
layout to blank widget
self.window.button.layout.addWidget(self.window.blank) #adding button
layout
self.window.button.layout.addWidget(self.window.button) #adding button
layout

self.window.button.clicked.connect(self.window.close) #when button is
clicked, window is hidden
self.window.layout.addWidget(self.window.tree) #adding tree list to layout
self.window.layout.addWidget(self.window.buttonWidget) #adding button to
layout
self.window.tree.itemDoubleClicked.connect(onOpen)#when item double clicked
```

In reference to lines 47 to 116 of the GUI in the section appendix, `openWindow()` is a function that is called when the user selects the "Open Pond" option from the menu bar. Two functions are included within the function (`onOpen()` and `tree()`).

It was necessary make this section of the code a function as it can be called repeatedly throughout the running of the program by the user. Similarly, `onOpen()` is only called when the user has selected a pond to open.

Looking at the `tree()` function, I think there's no real reason for the section of code to be a function but I decided to separate it to make it easier to distinguish and understand.

3.2 Function – savePond()

```
def savePond():
    try:
        self.mainPond.updatePond(db,cursor)
    except:
        self.errorWindow=QWidget()
        self.errorWindow.setWindowTitle("Error")
        self.errorWindow.setFixedWidth(250)
        self.errorWindow.setFixedHeight(150)
        self.errorWindow.show()
        self.errorWindow.layout = QVBoxLayout()
        self.errorWindow.setLayout(self.errorWindow.layout)
        self.errorWindow.label=QLabel("You must open a pond before it can
be saved")
        self.errorWindow.label.setWordWrap(True)
        self.errorWindow.widget=QWidget()
        self.errorWindow.widgetLayout=QGridLayout()
        self.errorWindow.widget.setLayout(self.errorWindow.widgetLayout)
        self.errorWindow.okay=QPushButton("OK")
        self.errorWindow.okay.setFixedWidth(50)
        self.errorWindow.okay.clicked.connect(self.errorWindow.close)
        blank=QWidget()
        blank.setFixedWidth(100)
        blankTwo=QWidget()
        blankTwo.setFixedWidth(100)
        self.errorWindow.widgetLayout.addWidget(blank,0,2)
        self.errorWindow.widgetLayout.addWidget(blankTwo,0,0)
        self.errorWindow.widgetLayout.addWidget(self.errorWindow.okay,0,1)
        self.errorWindow.layout.addWidget(self.errorWindow.label)
        self.errorWindow.layout.addWidget(self.errorWindow.widget)
```

In reference to lines 18 to 45 in the GUI section of the appendix, the `savePond()` function is called when a user selects to save the pond from the file menu on the menu bar. I made this a function for 2 reasons – the first being that parameters cannot be given to a function acting in response to an event (see line 118 of the GUI section – `savePond()` is given to the trigger as an event but cannot contain parameters), the second reason being I was able to add an exception error to display an error message to the user should they try to save a pond when one is not open.

3.3 Function – saveFishTypes()

```
def saveFishTypes(db,cursor):
    newGold=Goldfish(db,cursor)
    newTench=Tench(db,cursor)
    newOrfe=Orfe(db,cursor)
    newKoi=Koi(db,cursor)
    newRudd=Rudd(db,cursor)
    newShark=Shark(db,cursor)
    try:
        newGold.save(db, cursor)
        newTench.save(db, cursor)
        newOrfe.save(db, cursor)
        newKoi.save(db, cursor)
        newRudd.save(db, cursor)
        newShark.save(db, cursor)
    except:
        pass
```

In reference to lines 171 to 186 in the fish module in the section appendix, this function is required to save the allowed fish types to the database when the program is opened.

I made this section of a code a function so that it can be accessed by the main module, GUI, when the program is first run and the database is created.

3.4 Class – Fish

```
class Fish:
    """A simulation of a generic fish"""
    def __init__(self, db, cursor):
        self._fishName=None
        self._fishType="fish"
        self._foodNeed=0
        self._growthRate=0
        self._fishSize=0
        self._hunger=0
        self._status="good"
        self._daysAlive=0
        self._growth=0
        self._maxDaysAlive=100
        self._fishID=None

    def getFishID(self):
        return self._fishID

    def setFishName(self,x):
        self._fishName=x

    def getFishName(self):
        return self._fishName

    def setFishType(self,x):
        self._fishType=x

    def getFishType(self):
        return self._fishType

    def setFoodNeed(self,x):
        self._foodNeed=x

    def getFoodNeed(self):
        return self._foodNeed

    def setGrowthRate(self,x):
        self._growthRate=x

    def getGrowthRate(self):
        return self._growthRate

    def setFishSize(self,x):
        self._setFishSize=x

    def getFishSize(self):
        return self._fishSize

    def setHunger(self,x):
        self._hunger=x

    def getHunger(self):
        return self._hunger

    def setStatus(self,growth,daysAlive):
        pass

    def getStatus(self):
        return self._status

    def setDaysAlive(self,x):
        self._daysAlive=x

    def getDaysAlive(self):
        return self._daysAlive

    def setGrowth(self,x):
        self._growth=x
    def getGrowth(self):
        return self._growth

    def grow(self,food):
        pass

    def saveFish(self, pondID, db, cursor):
        sql = """insert into fish(fishName, hunger, status, daysAlive, 
growth, fishType, pondID) values
                (
'{0}','{1}','{2}','{3}','{4}','{5}','{6}')""".format(self._fishName, 
self._hunger, self._status, self._daysAlive, self._growth, self._fishType, 
pondID)
        cursor.execute(sql)
        cursor.execute("""select last_insert_rowid()""")
        self._fishID = cursor.fetchall()[0][0]
        db.commit()
```

In reference to lines 3 to 84 in the fish module in the section appendix, this is the base class for all the different fish objects. There are 6 different types of fish that can be added to my pond, so there are 6 corresponding fish objects. I made a base fish class to be inherited by the other fish classes in order to stop unnecessary duplication of code, as all the fish class methods would need to added to the 6 different fish classes.

Fish is a class as all fish objects need specific attributes about them to be stored, so they can be added to the database correctly.

3.5 Class – Pond

```
class Pond:
    """A simulation of a pond"""
    def __init__(self, width, length, depth, temp, waterLevel, pump,
filter, pondID, days):
        self._pondWidth=width
        self._pondLength=length
        self._pondDepth=depth
        self._waterTemp=temp
        self._waterLevel=waterLevel
        self._pump=pump
        self._filter=filter
        self._pondID=pondID
        self._days=days

    def setPondID(self,x):
        self._pondID=x

    def getPondID(self):
        return self._pondID

    def setPondWidth(self, x): #where x is the desired pond width
        self._pondWidth=x

    def getPondWidth(self):
        return self._pondWidth

    def setPondLength(self, x):
        self._pondLength=x

    def getPondLength(self):
        return self._pondLength

    def setPondDepth(self,x):
        self._pondDepth=x

    def getPondDepth(self):
        return self._pondDepth

    def setWaterLevel(self,x):
        self._waterLevel=x

    def getWaterLevel(self):
        return self._waterLevel

    def setWaterTemp(self, x):
        self._waterTemp=x

    def getWaterTemp(self):
        return self._waterTemp

    def setPump(self, x): #where x is the pump object
        self._pump=x

    def getPump(self):
        return self._pump

    def setFilter(self, x):
        self._filter=x

    def getFilter(self):
        return self._filter

    def setDays(self, x):
        self._days=x

    def getDays(self):
        return self._days

    def addFish(self, db, cursor,choice,name):
        def chooseFish(db, cursor,choice):
            if choice=="Goldfish":
                newFish=fish.Goldfish(db, cursor)
            elif choice=="Tench":
                newFish=fish.Tench(db,cursor)
            elif choice=="Orfe":
                newFish=fish.Orfe(db,cursor)
            elif choice=="Koi":
                newFish=fish.Koi(db,cursor)
            elif choice=="Rudd":
                newFish=fish.Rudd(db,cursor)
            elif choice=="Shark":
                newFish=fish.Shark(db,cursor)
            return newFish

        newFish=chooseFish(db, cursor,choice)
        newFish.setFishName(name)
        newFish.saveFish(self._pondID, db, cursor)

    def removeFish(self,db, cursor):
        pondID=pond.getPondID()
        fishTup=pondSimDatabase.fetchFish(db, cursor, pondID)
        correct=False

        #displaying list of fish in pond

print('|{0:^10}|{1:^10}|{2:^15}|{3:^15}|{4:^15}|{5:^15}|{6:^10}|'.format('F
ishID','fishName','Hunger','Status','daysAlive','growth','FishType'))
        for each in range(0,len(fishTup)):
            print('{0:^12} {1:^10} {2:^10} {3:^20} {4:^10} {5:^20}
{6:^5}'.format(fishTup[each][0],fishTup[each][1],fishTup[each][2],fishTup[e
ach][3],fishTup[each][4],fishTup[each][5],fishTup[each][6]))
        print('')

        try:
            while correct==False:
                IDchoice=int(input('Enter the ID of the fish you wish to
remove: '))
                if IDchoice>len(fishTup)or IDchoice<0:
                    print("That ID was not found on the list")
                else:
                    correct=True
                    pondSimDatabase.deleteFish(db, cursor, IDchoice)
        except:
            print("That was not a valid ID - please enter a number")


    def removePlant(self,db,cursor):
        pondID=pond.getPondID()
        plantsTup=pondSimDatabase.fetchPlants(db, cursor, pondID)
        correct=False


print('|{0:^10}|{1:^10}|{2:^15}|{3:^15}|{4:^15}|{5:^15}|'.format('PlantID',
'PlantType','Status','LightNeed','daysAlive','growth'))
        for each in range(0,len(plantsTup)):
            print('{0:^12} {1:^10} {2:^10} {3:^20} {4:^10}
{5:^20}'.format(plantsTup[each][0],plantsTup[each][1],plantsTup[each][2],pl
antsTup[each][3],plantsTup[each][4],plantsTup[each][5]))
        print('')

        try:
            while correct==False:
                IDchoice=int(input('Enter the ID of the plant you wish to
remove: '))
                if IDchoice>len(plantTup)or IDchoice<0:
                    print("That ID was not found on the list")
                else:
                    correct=True
                    pondSimDatabase.deletePlant(db, cursor, IDchoice)
        except:
            print("That was not a valid ID - please enter a number")

    def addPlant(self, db, cursor,choice):
        def choosePlant(db,cursor,choice):
            if choice=="Rooted Floater":
                newPlant=plant.RootedFloater(db,cursor)
            elif choice=="True Floater":
                newPlant=plant.TrueFloater(db,cursor)
            elif choice=="Emergent":
                newPlant=plant.Emergent(db,cursor)
            elif choice=="Submersed":
                newPlant=plant.Submersed(db,cursor)
            return newPlant
        newPlant=choosePlant(db,cursor,choice)
        newPlant.savePlant(self._pondID, db, cursor)

    def editPond(self):
        waterTemp=input("Water Temperature:")
        self.setWaterTemp(waterTemp)

        pondWidth=input("Pond Width:")
        self.setPondWidth(pondWidth)

        pondLength=input("Pond Length:")
        self.setPondLength(pondLength)

        pondDepth=input("Pond Depth:")
        self.setPondDepth(pondDepth)

        self.setWaterLevel(int(pondDepth)-0.5)

    def savePond(self,db,cursor):
        #saving pond
        try:
            self._pumpID=self._pump.getPumpID()
        except:
            self._pumpID=None

        try:
            self._filterID=self._filter.getFilterID()
        except:
            self._filterID=None

        sql = """insert into pond(pondDepth, pondLength, pondWidth,
waterTemp, waterLevel, pumpID, filterID, days) values
                 (
'{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}')""".format(self._pondDepth,
self._pondLength, self._pondWidth, self._waterTemp, self._waterLevel,
self._pump, self._filter, self._days)
        cursor.execute(sql)
        cursor.execute("""select last_insert_rowid()""")
        self._pondID = cursor.fetchall()[0][0]
        db.commit()

    def updatePond(self,db,cursor):
        try:
            self._pumpID=self.getPump()
        except:
            self._pumpID=None

        try:
            self._filterID=self.getFilter()
        except:
            self._filterID=None
        try:
            tup=(self._pondDepth, self._pondLength, self._pondWidth,
self._waterTemp, self._waterLevel, self._days, self._pumpID,
self._filterID, self._pondID)
            sql = """update pond
                    set pondDepth=?, pondLength=?, pondWidth=?,
waterTemp=?, waterLevel=?, days=?, pumpID=?, filterID=?
                    where pondID=?"""
            cursor.execute(sql,tup)
            db.commit()
        except:
            pass
```

In reference to lines 9 to 206 in the pond module in the section, appendix, this is a class for defining and accessing ponds. Clearly this section needed to be a class as I previously identified ponds as objects which require specific attribues.

As well as the getter and setter methods, the pond class contains several methods; `updatePond()`,`savePond()`,`editPond()`,`addPlant()`, `removePlant()`,`addFish()`, `removeFish()`. These methods are all part of the pond class as they all require access to the pond object to work. The methods `addPlant()`, `removePlant()`,`addFish()`, `removeFish()` could have been including in the fish and plant modules instead, but I decided to include them with the pond module as it seemed easier to think of the fish and plants belonging to a pond.

The `incrementDay()` function (lines 433 to 520) could also have been included in the pond module.

4. Variable Listing

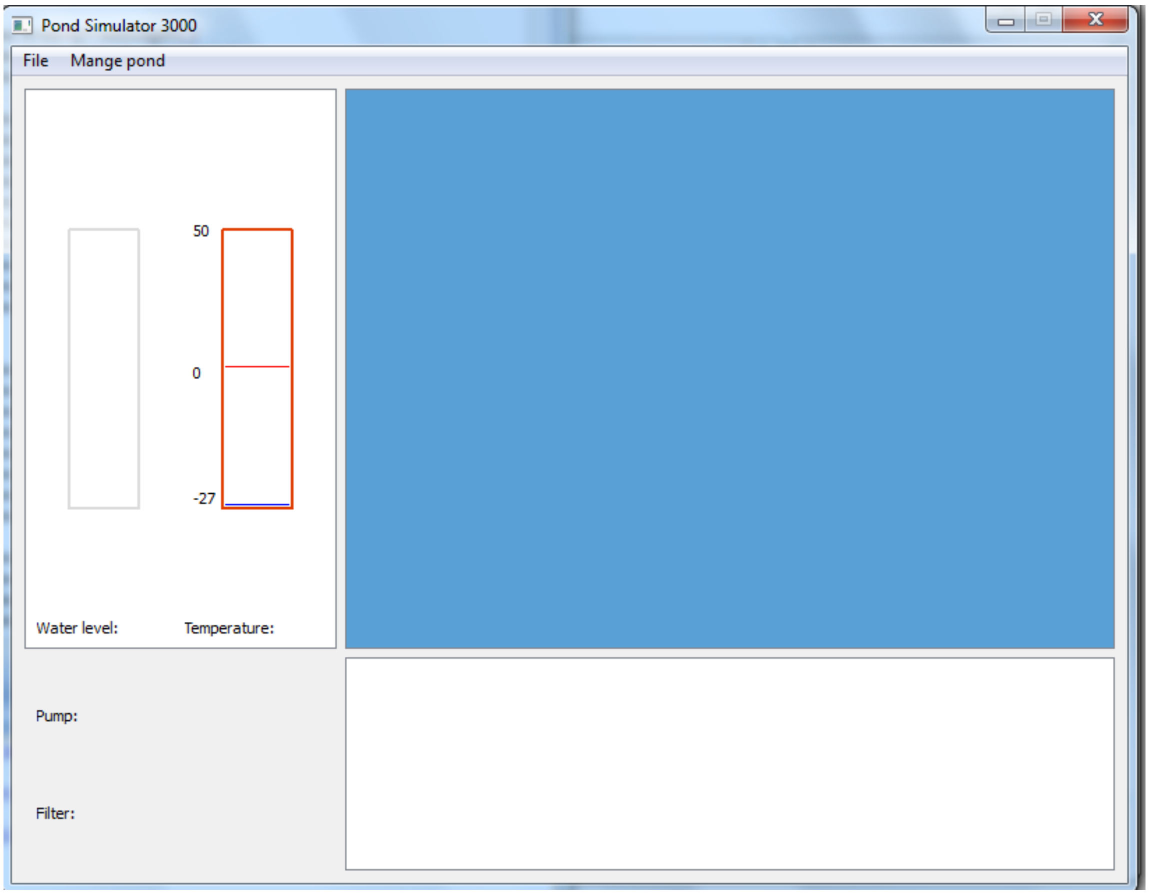
The following is a table containing a complete list of variables in my program. The data dictionary can be found in Design Section 10.4.

Variable Name	**Purpose**	**Location in code**
`self.errorWindow`	Blank widget for error message	gui.py, line 22
`self.errorWindow.layout`	Layout for blank widget	Gui.py, line 27
`self.errorWindow.label`	Label displaying error message to be added to layout	Gui.py, 29
`self.errorWindow.okay`	Push button to be added to layout under label	Gui.py, line 34
`self.selected`	Getting the selected pond to open	Gui.py, line 51
`pondID`	Getting the pond id from the selected pond	Gui.py, line 52
`depth`	Getting depth from selected pond	Gui.py, line 53
`length`	Getting length from selected pond	Gui.py, line 54
`width`	Getting width from selected pond	Gui.py, line 55
`temp`	Getting temp from selected pond	Gui.py, line 56
`level`	Getting water level from selected pond	Gui.py, line 57
`pump`	Getting pump ID from selected pond	Gui.py, line 58
`filter`	Getting filter ID from selected pond	Gui.py, line 59
`days`	Getting days from selected pond	Gui.py, line 60
`self.mainPond`	The pond currently open in the program	Gui.py, lines 61, 282
`results`	A list of all the ponds in the database	Gui.py, line 66
`self.window.treeList`	A QTreeWidgetItem for displaying the list of ponds in the open pond window	Gui.py, line 68
`self.window`	Blanket widget window	Gui.py, line 80, 119, 292, 447, 481, 611, 650, 817, 841
`db`	Allows me to connect to the database	Gui.py, line 1292
`cursor`	Allows me to execute sql methods in my code using the `cursor.execute()` method	Gui.py, line 1293
`application`	My actual application object that everything is built on	Gui.py, line 1298
`mainWindow`	The main window of my program	Gui.py, line 1299
`goldfishTotal`	The total number of	Gui.py, line 1215, 1224

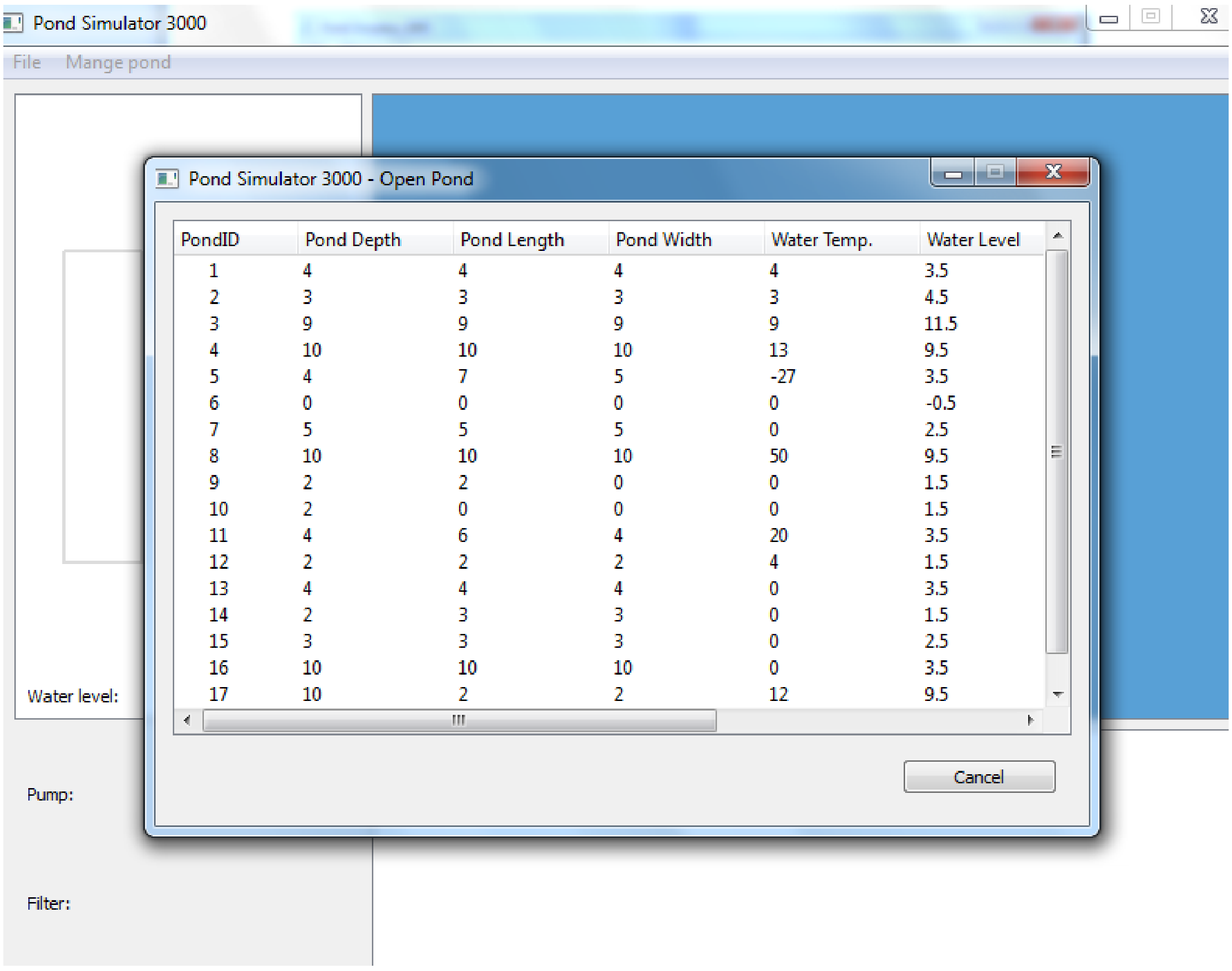
	goldfish in the pond, to be displayed to the user in the main window	
`tenchTotal`	The total number of tench in the pond, to be displayed to the user in the main window	Gui.py, line 1216, 1226
`orfeTotal`	The total number of orfe in the pond, to be displayed to the user in the main window	Gui.py, line 1217, 1228
`koiTotal`	The total number of koi in the pond, to be displayed to the user in the main window	Gui.py, line 1218, 1230
`ruddTotal`	The total number of rudd in the pond, to be displayed to the user in the main window	Gui.py, line 1219, 1232
`sharkTotal`	The total number of shark in the pond, to be displayed to the user in the main window	Gui.py, line 1220, 1234
`emergentTotal`	The total number of emergent plants in the pond, to be displayed to the user in the main window	Gui.py, line 1257, 1264
`rootedTotal`	The total number of rooted floating plants in the pond, to be displayed to the user in the main window	Gui.py, line 1258, 1266
`trueTotal`	The total number of true floating plants in the pond, to be displayed to the user in the main window	Gui.py, line 1259, 1268
`submesedTotal`	The total number of submersed plants in the pond, to be displayed to the user in the main window	Gui.py, line 1260, 1270

5. System Evidence

5.1 User Interface

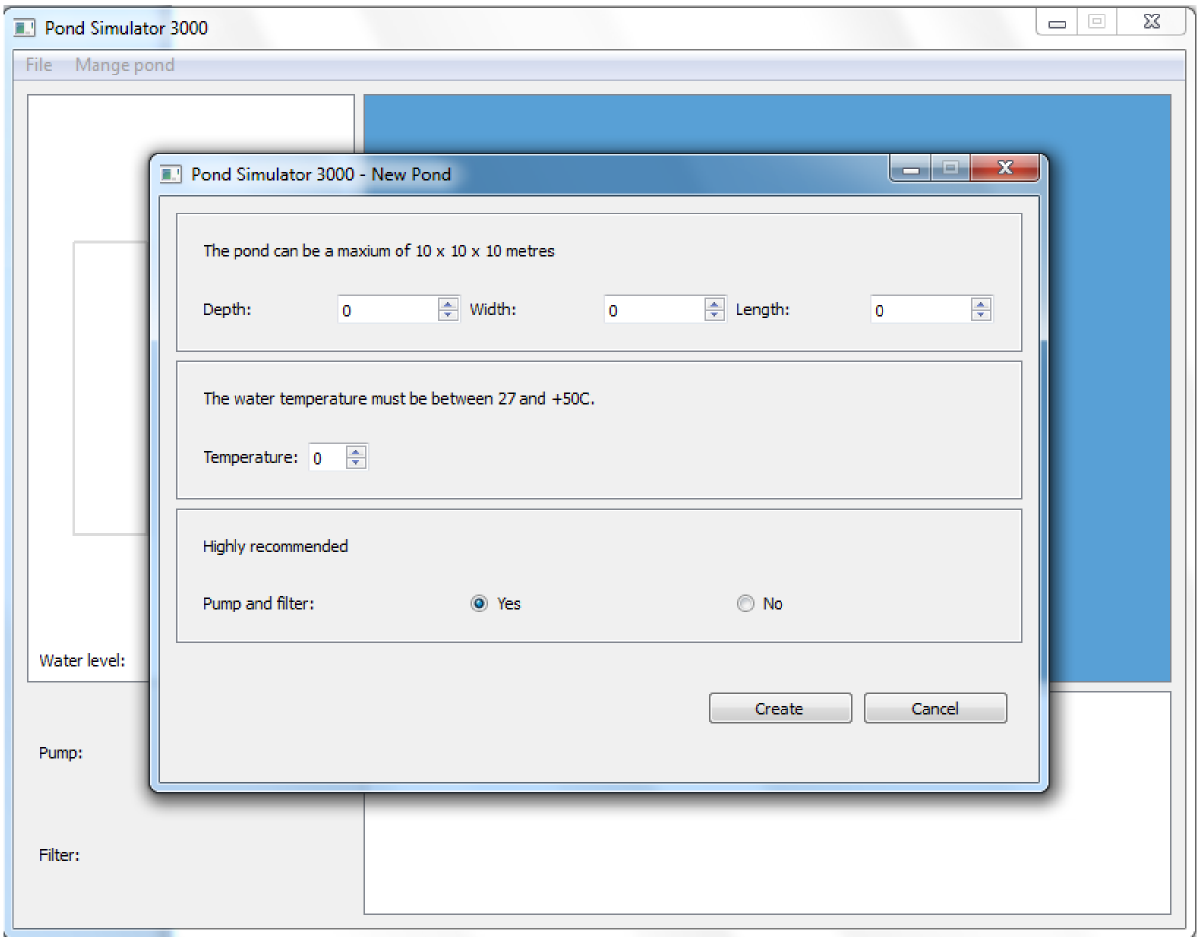


The main window of the program as it appears on opening.



PondID	Pond Depth	Pond Length	Pond Width	Water Temp.	Water Level
1	4	4	4	4	3.5
2	3	3	3	3	4.5
3	9	9	9	9	11.5
4	10	10	10	13	9.5
5	4	7	5	-27	3.5
6	0	0	0	0	-0.5
7	5	5	5	0	2.5
8	10	10	10	50	9.5
9	2	2	0	0	1.5
10	2	0	0	0	1.5
11	4	6	4	20	3.5
12	2	2	2	4	1.5
13	4	4	4	0	3.5
14	2	3	3	0	1.5
15	3	3	3	0	2.5
16	10	10	10	0	3.5
17	10	2	2	12	9.5

The open pond window



The pond can be a maxium of 10 x 10 x 10 metres

Depth: 0 Width: 0 Length: 0

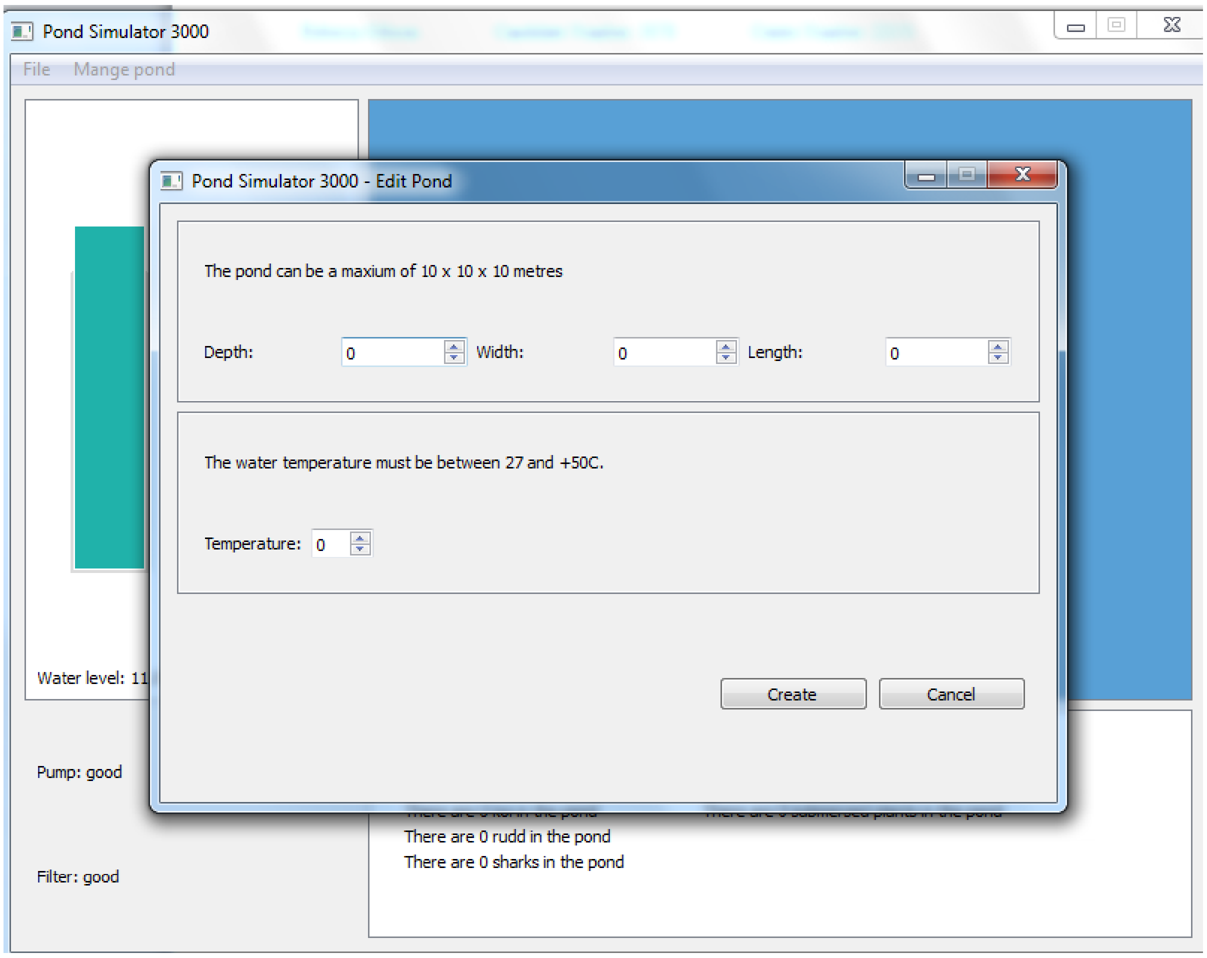
The water temperature must be between 27 and +50C.

Temperature: 0

Highly recommended

Pump and filter: [✓ Yes] No

New pond window



The pond can be a maxium of 10 x 10 x 10 metres

Depth: 0 Width: 0 Length: 0

The water temperature must be between 27 and +50C.

Temperature: 0

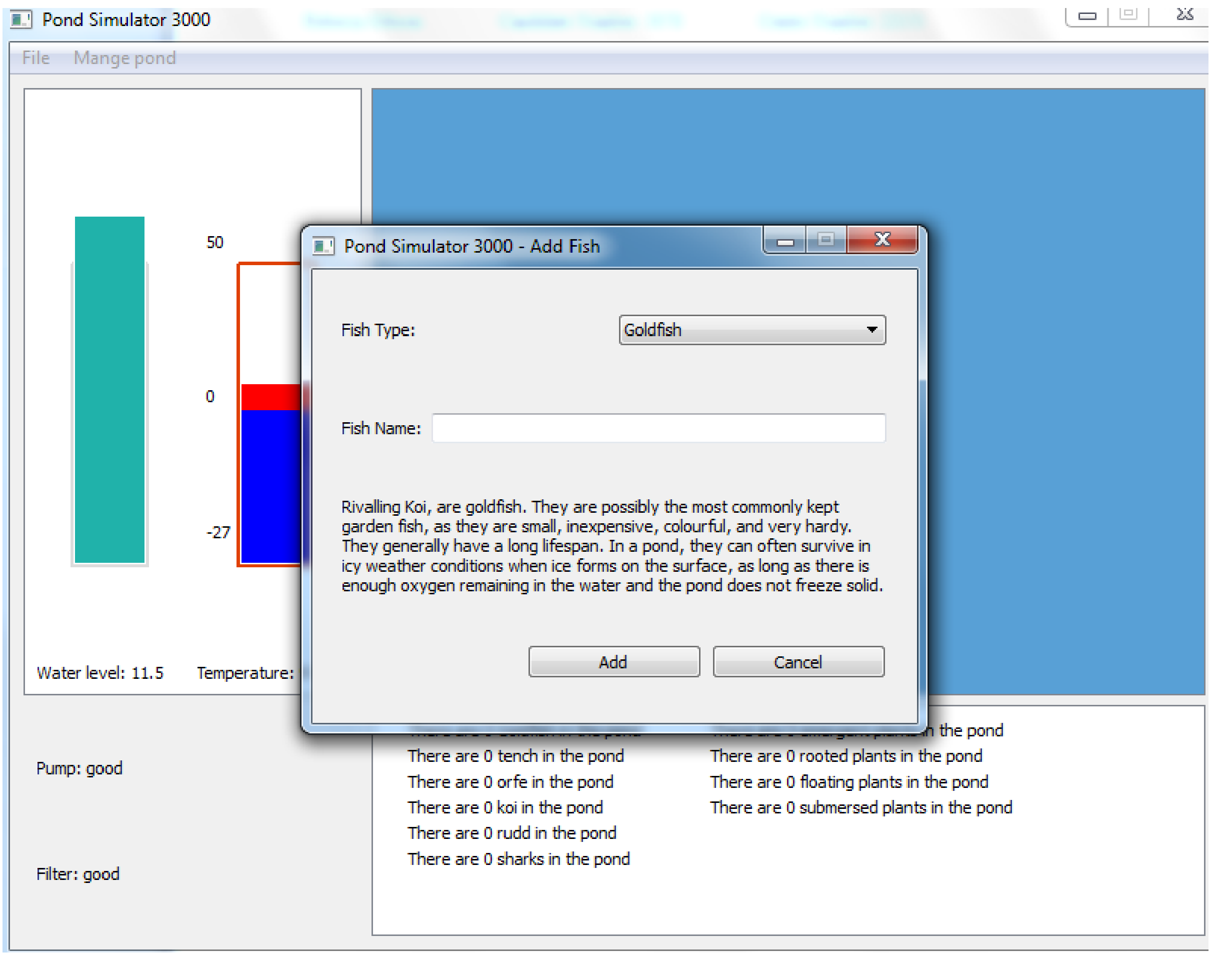
Water level: 11

Pump: good

Filter: good

There are 0 rudd in the pond
There are 0 sharks in the pond

Edit pond window



Fish Type: Goldfish

Fish Name:

Rivalling Koi, are goldfish. They are possibly the most commonly kept garden fish, as they are small, inexpensive, colourful, and very hardy. They generally have a long lifespan. In a pond, they can often survive in icy weather conditions when ice forms on the surface, as long as there is enough oxygen remaining in the water and the pond does not freeze solid.

Water level: 11.5 Temperature:

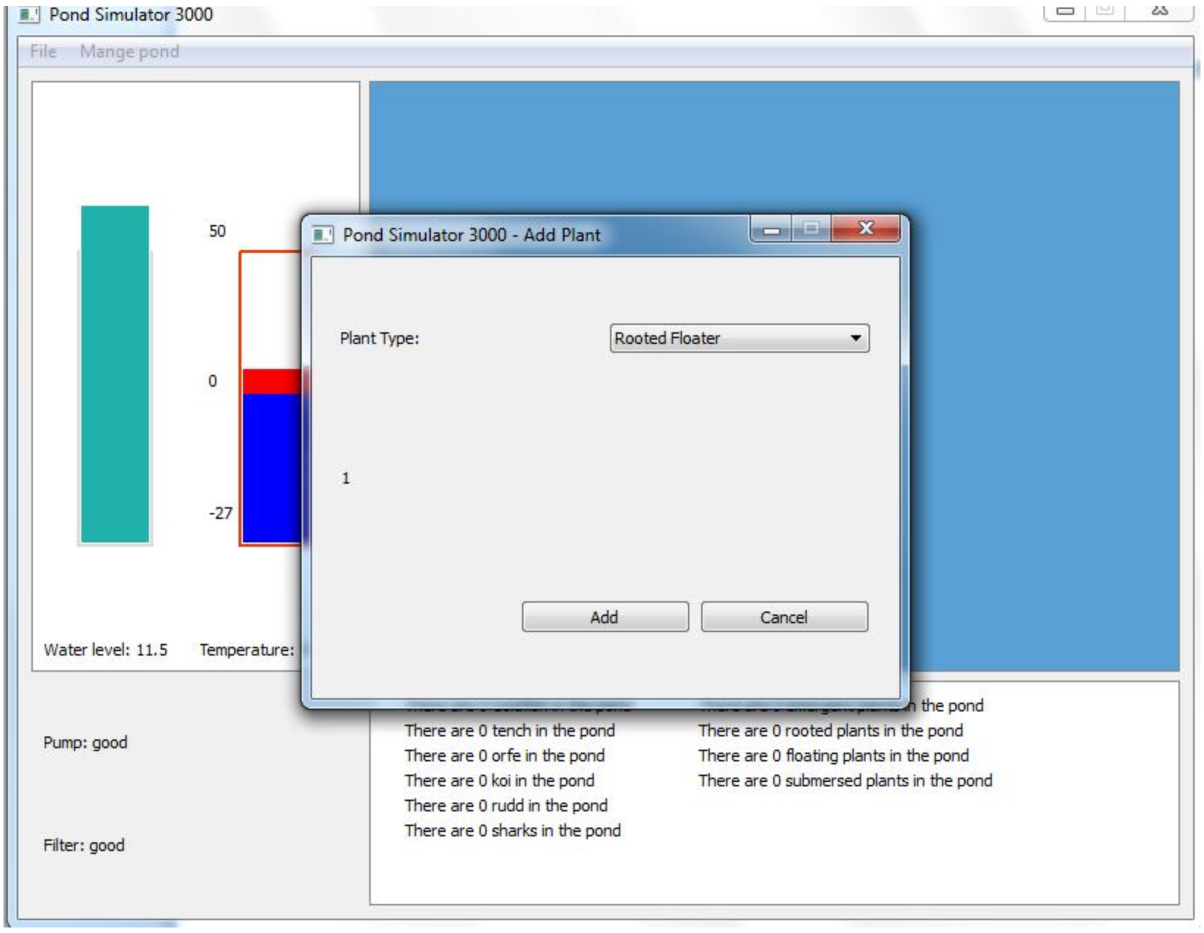
Pump: good

Filter: good

There are 0 tench in the pond
There are 0 orfe in the pond
There are 0 koi in the pond
There are 0 rudd in the pond
There are 0 sharks in the pond

the pond
There are 0 rooted plants in the pond
There are 0 floating plants in the pond
There are 0 submersed plants in the pond

Add fish window



Plant Type: Rooted Floater

1

Water level: 11.5 Temperature:

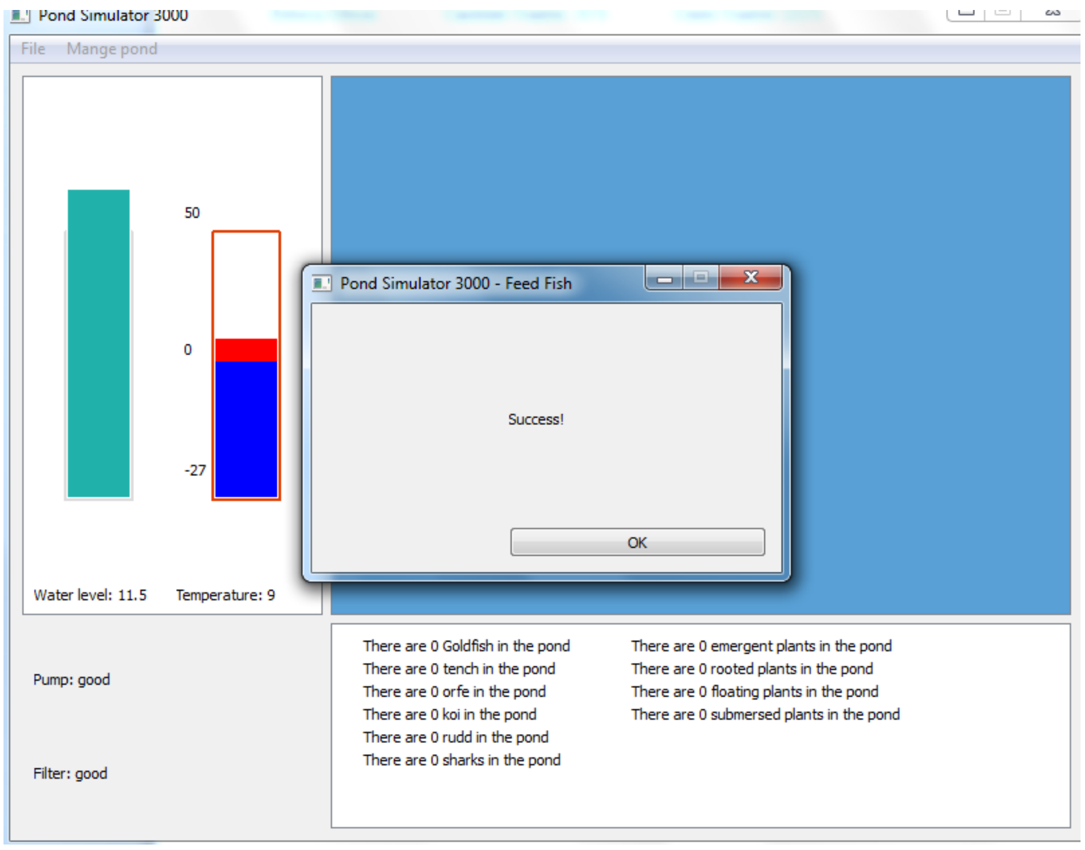
Pump: good

Filter: good

There are 0 tench in the pond
There are 0 orfe in the pond
There are 0 koi in the pond
There are 0 rudd in the pond
There are 0 sharks in the pond

the pond
There are 0 rooted plants in the pond
There are 0 floating plants in the pond
There are 0 submersed plants in the pond

Add plant window



Success!

Water level: 11.5 Temperature: 9

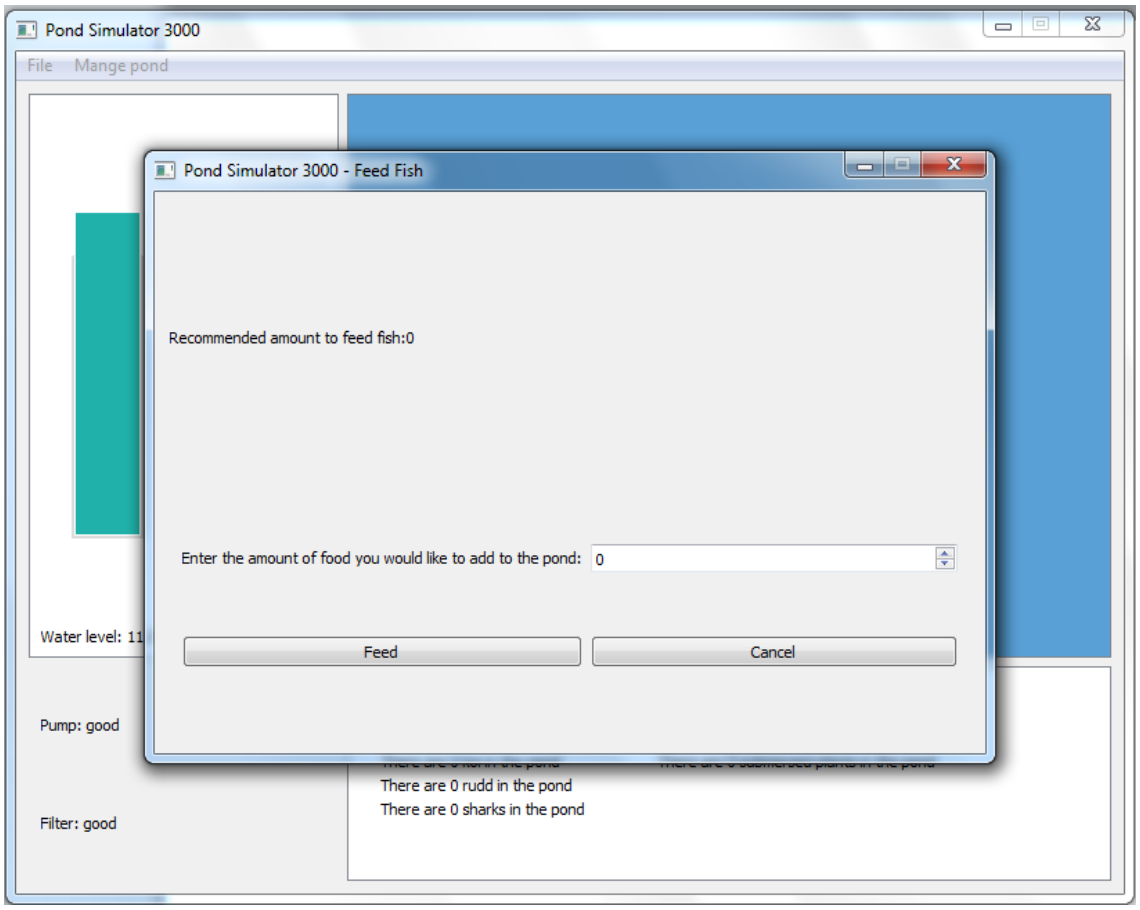
Pump: good

Filter: good

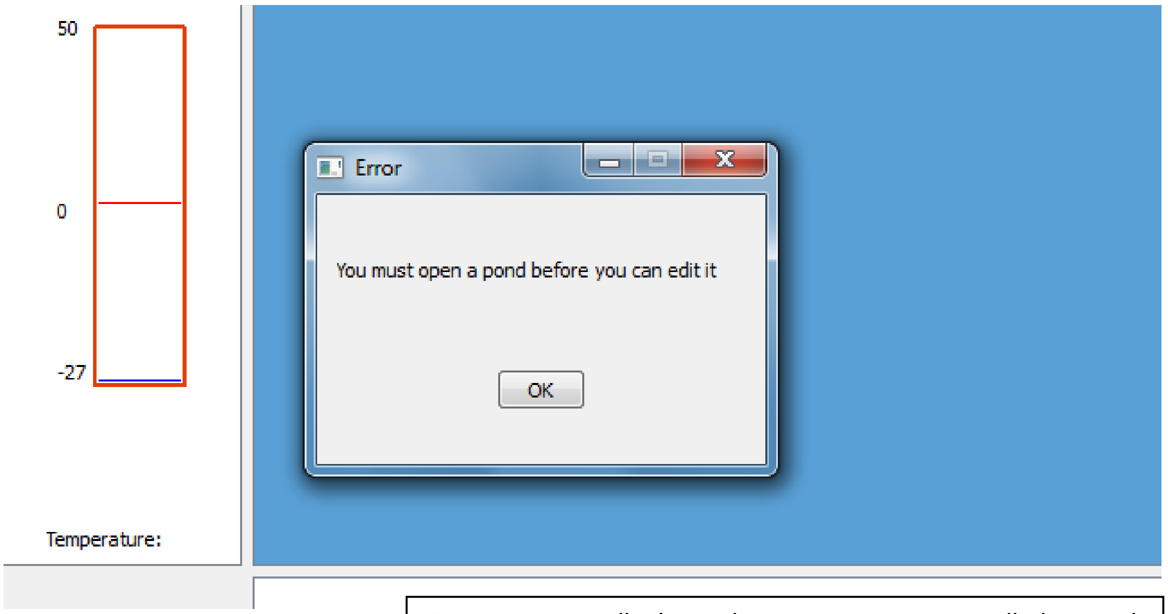
There are 0 Goldfish in the pond
There are 0 tench in the pond
There are 0 orfe in the pond
There are 0 koi in the pond
There are 0 rudd in the pond
There are 0 sharks in the pond

There are 0 emergent plants in the pond
There are 0 rooted plants in the pond
There are 0 floating plants in the pond
There are 0 submersed plants in the pond

Message telling user fish have been fed successfully

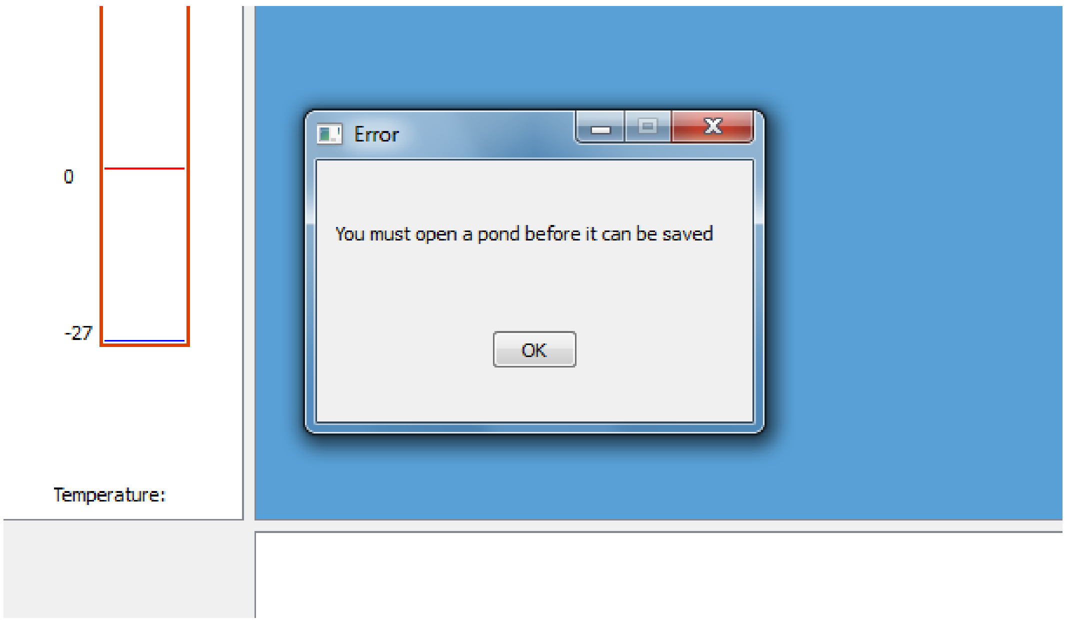

Pond Simulator 3000
File
Mange pond
Pond Simulator 3000 - Feed Fish
Recommended amount to feed fish:0
Enter the amount of food you would like to add to the pond:
0
Feed
Cancel
Water level: 11
Pump: good
Filter: good
There are 0 rudd in the pond
There are 0 sharks in the pond

Feed fish manually window



You must open a pond before you can edit it

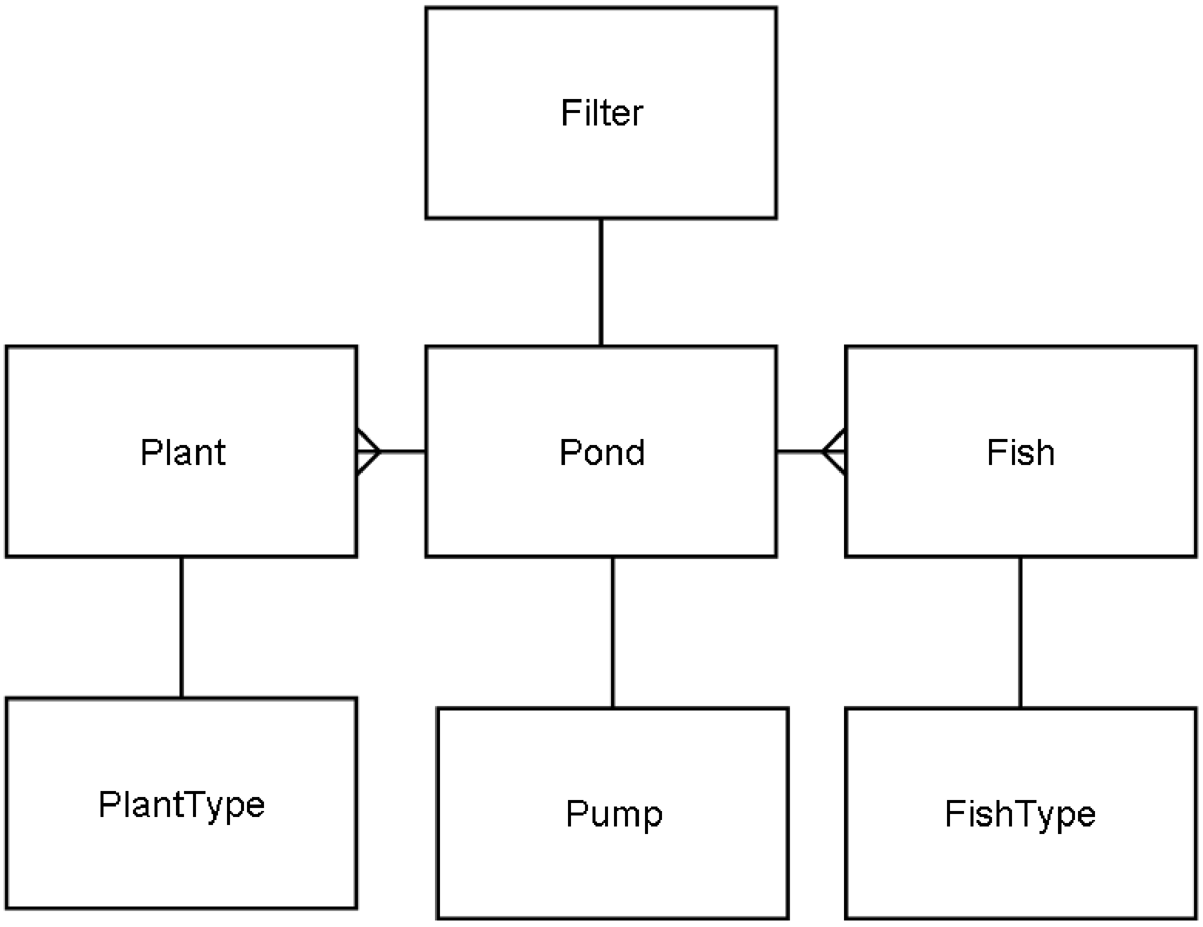
Error message displays when user attempts to edit the pond when one is not selected.



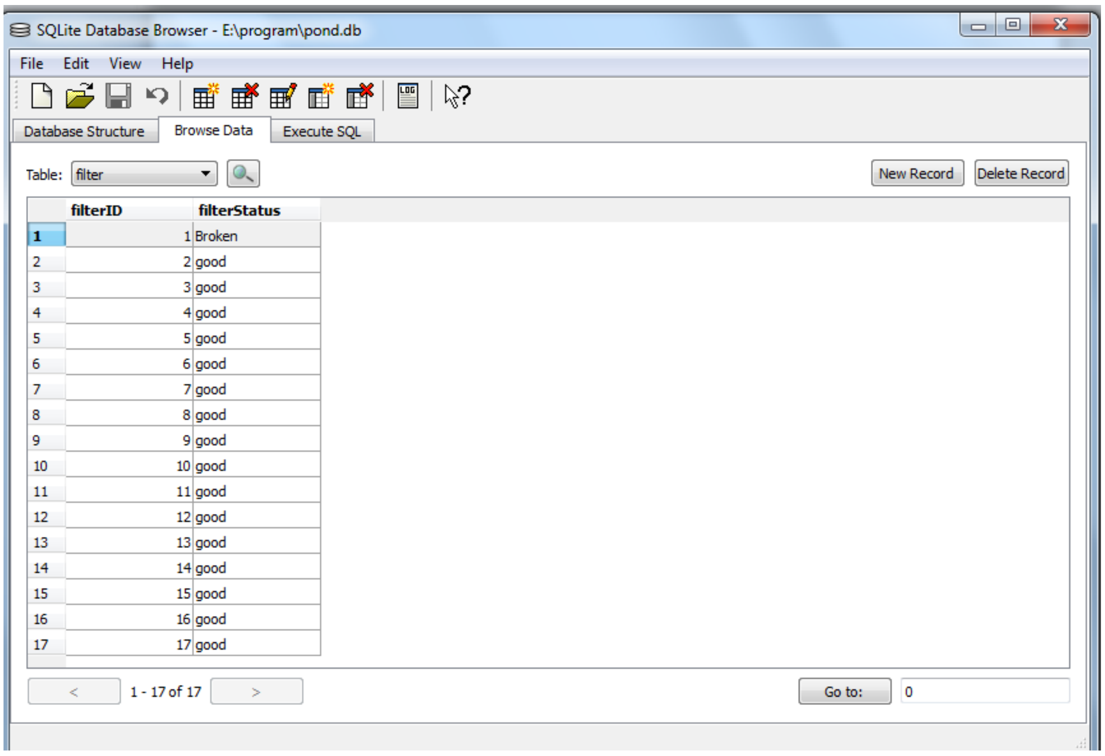
You must open a pond before it can be saved

Error message displays when use attempts to save the pond when one is not selected.

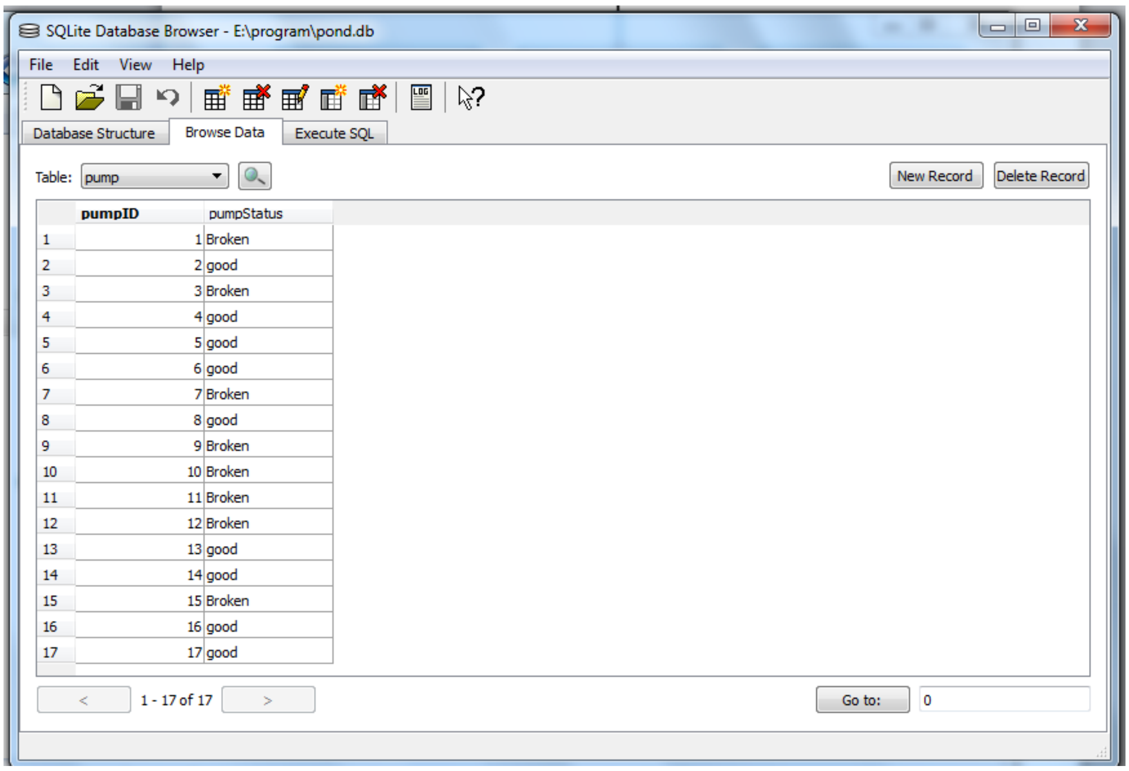
5.2 ER Diagram



5.3 Database Table Views



	filterID	filterStatus
1	1	Broken
2	2	good
3	3	good
4	4	good
5	5	good
6	6	good
7	7	good
8	8	good
9	9	good
10	10	good
11	11	good
12	12	good
13	13	good
14	14	good
15	15	good
16	16	good
17	17	good



	pumpID	pumpStatus
1	1	Broken
2	2	good
3	3	Broken
4	4	good
5	5	good
6	6	good
7	7	Broken
8	8	good
9	9	Broken
10	10	Broken
11	11	Broken
12	12	Broken
13	13	good
14	14	good
15	15	Broken
16	16	good
17	17	good

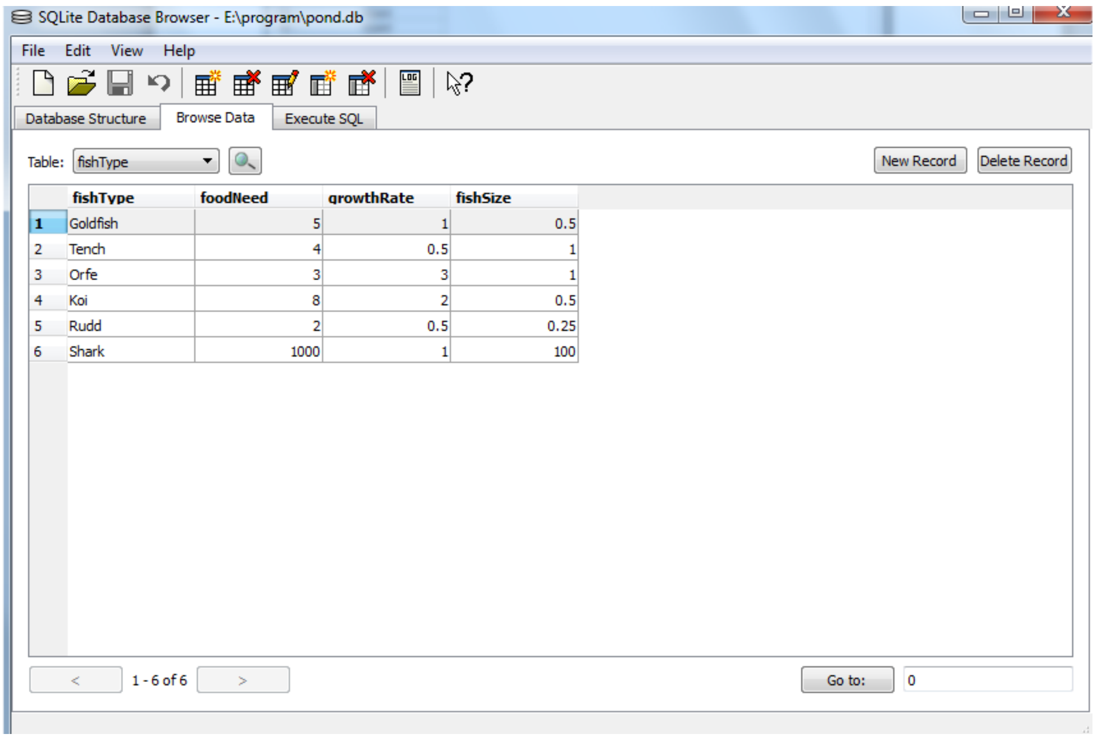


Table: fishType

	fishType	foodNeed	growthRate	fishSize
1	Goldfish	5	1	0.5
2	Tench	4	0.5	1
3	Orfe	3	3	1
4	Koi	8	2	0.5
5	Rudd	2	0.5	0.25
6	Shark	1000	1	100

1 - 6 of 6

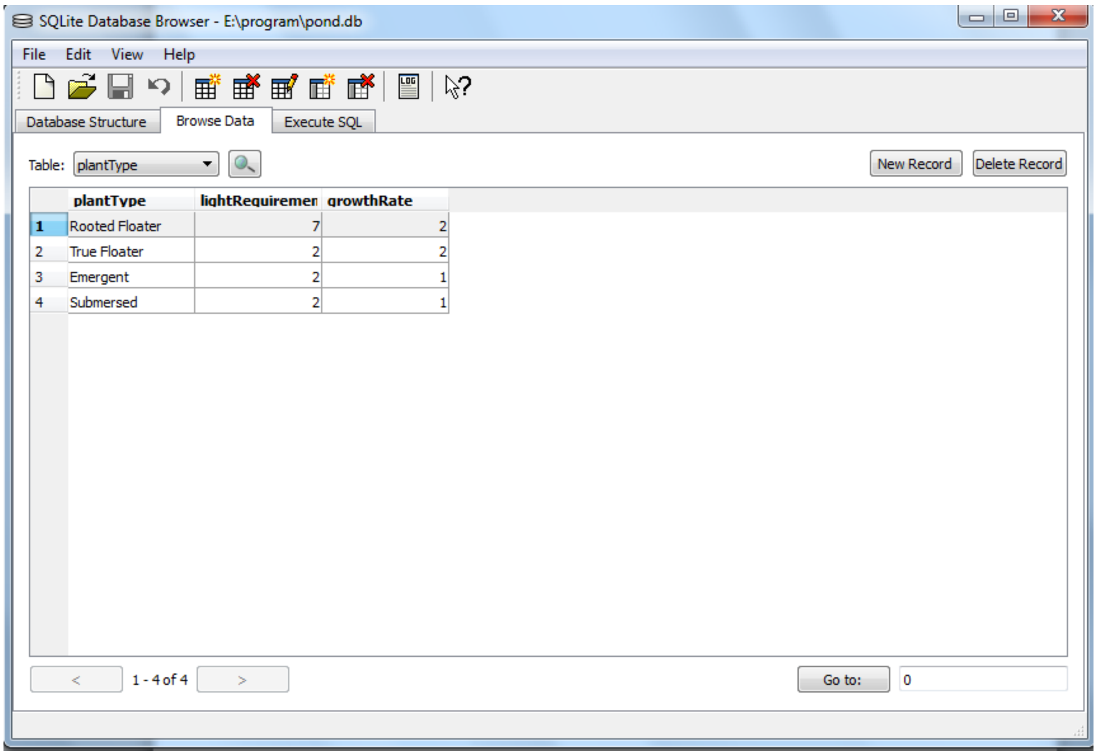
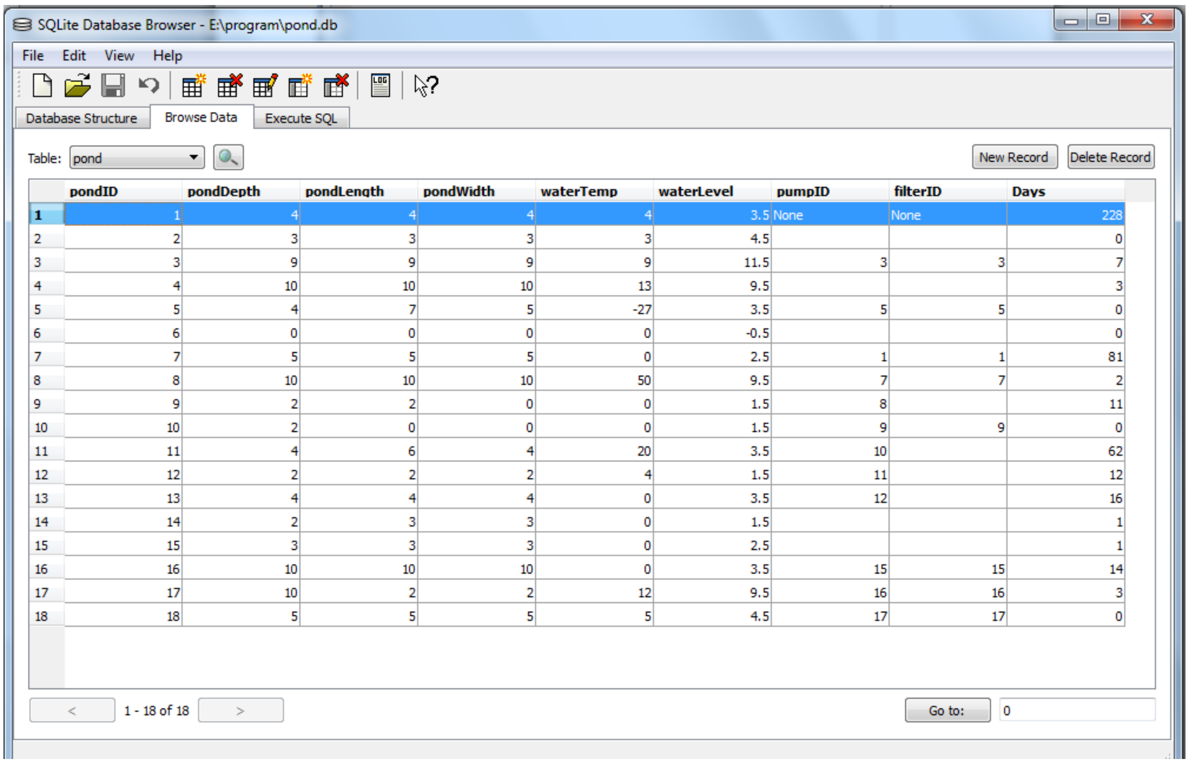


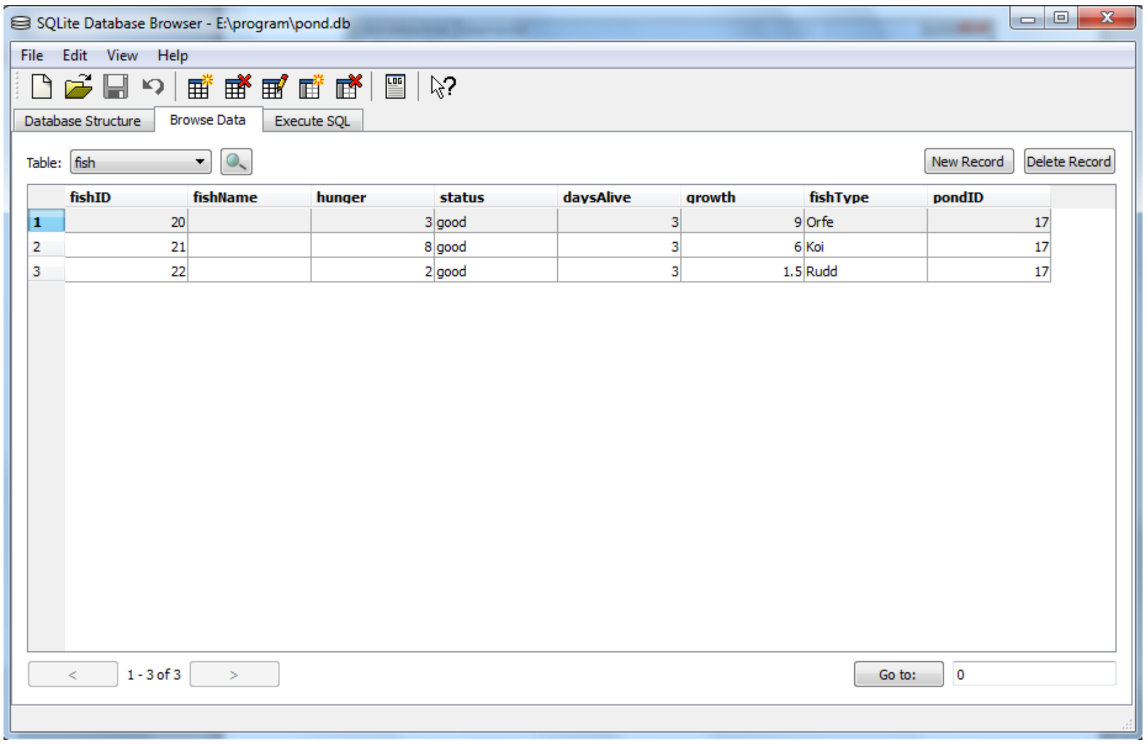
Table: plantType

	plantType	lightRequiremen	growthRate
1	Rooted Floater	7	2
2	True Floater	2	2
3	Emergent	2	1
4	Submersed	2	1

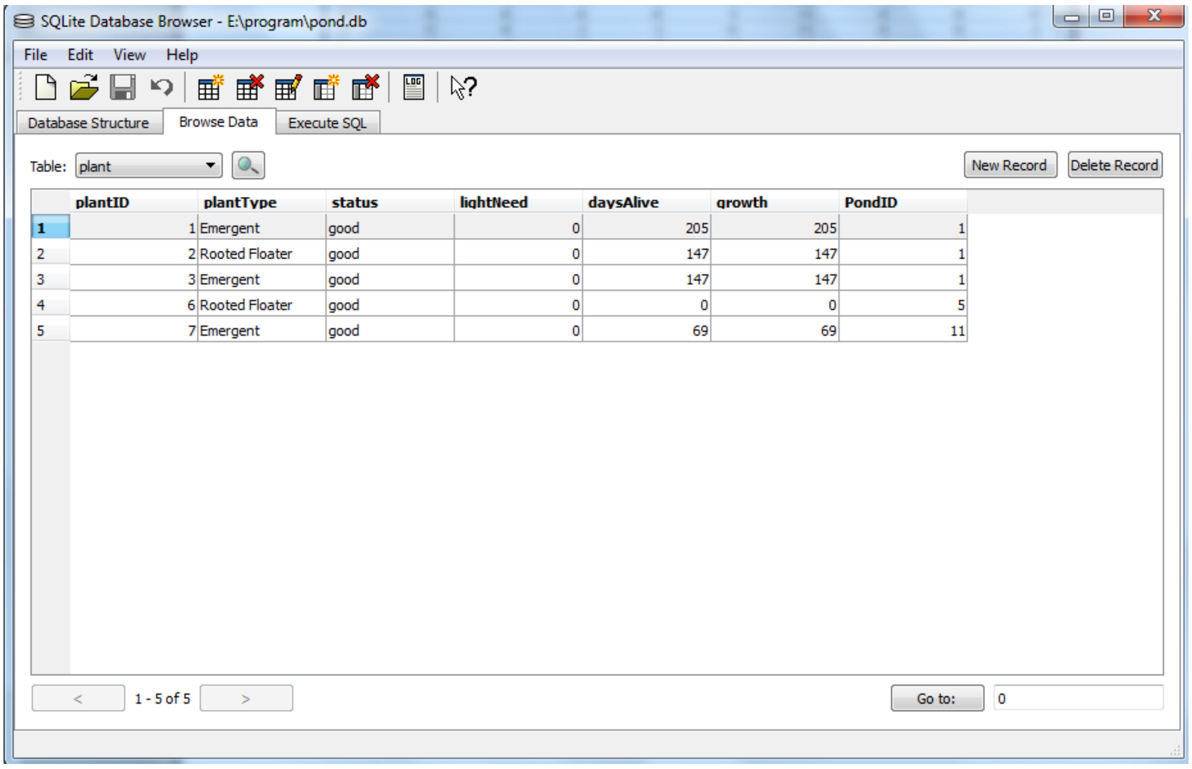
1 - 4 of 4



	pondID	pondDepth	pondLength	pondWidth	waterTemp	waterLevel	pumpID	filterID	Days
1	1	4	4	4	4	3.5	None	None	228
2	2	3	3	3	3	4.5			0
3	3	9	9	9	9	11.5	3	3	7
4	4	10	10	10	13	9.5			3
5	5	4	7	5	-27	3.5	5	5	0
6	6	0	0	0	0	-0.5			0
7	7	5	5	5	0	2.5	1	1	81
8	8	10	10	10	50	9.5	7	7	2
9	9	2	2	0	0	1.5	8		11
10	10	2	0	0	0	1.5	9	9	0
11	11	4	6	4	20	3.5	10		62
12	12	2	2	2	4	1.5	11		12
13	13	4	4	4	0	3.5	12		16
14	14	2	3	3	0	1.5			1
15	15	3	3	3	0	2.5			1
16	16	10	10	10	0	3.5	15	15	14
17	17	10	2	2	12	9.5	16	16	3
18	18	5	5	5	5	4.5	17	17	0



	fishID	fishName	hunger	status	daysAlive	growth	fishType	pondID
1	20		3	good	3	9	Orfe	17
2	21		8	good	3	6	Koi	17
3	22		2	good	3	1.5	Rudd	17



	plantID	plantType	status	lightNeed	daysAlive	growth	PondID
1	1	Emergent	good	0	205	205	1
2	2	Rooted Floater	good	0	147	147	1
3	3	Emergent	good	0	147	147	1
4	6	Rooted Floater	good	0	0	0	5
5	7	Emergent	good	0	69	69	11

5.4 Database SQL

5.4.1 Pond Database Table

```
create table pond(
pondID integer,
pondDepth integer,
pondLength integer,
pondWidth integer,
waterTemp integer,
waterLevel integer,
pumpID integer,
filterID integer,
Days integer,
primary key(pondID),
foreign key(pumpID) references pump(pumpID) on update cascade on delete
restrict,
foreign key(filterID) references filter(filterID) on update cascade on
delete restrict)
```

5.4.2 Fish Type Database Table

```
create table fishType(
fishType text,
foodNeed integer,
growthRate integer,
fishSize integer,
primary key(fishType))
```

5.4.3 Fish Database Table

```
create table fish(
fishID integer,
fishName text,
hunger integer,
status text,
daysAlive integer,
growth integer,
fishType text,
pondID integer,
primary key(fishID),
foreign key(fishType) references fishType(fishType),
foreign key(pondID) references pond(pondID))
```

5.4.4 Plant Type Database Table

```
create table plantType(
plantType text,
lightRequirement integer,
growthRate integer,
primary key(plantType))
```

5.4.5 Plant Database Table

```
create table plant(
plantID integer,
plantType text,
status text,
lightNeed integer,
daysAlive integer,
growth integer,
PondID integer,
primary key(plantID),
foreign key(plantType) references plantType(plantType),
foreign key(pondID) references pondID(pondID))
```

5.4.6 Pump Database Table

```
create table pump(
pumpID integer,
pumpStatus text,
primary key (pumpID))
```

5.4.7 Filter Database Table

```
create table filter(
filterID integer,
filterStatus text,
primary key (filterID))
```

5.5 SQL Queries

5.5.1 Fetch all Ponds

```
select *
from pond
```

Reference: getPond function in pondSimDatabase, lines 4 to 9

5.5.1 Fetch Pond where ID is user selection

```
select *
from pond
where pondID=?
```

Reference: fetchPond function in pondSimDatabase, lines 11 to 18

5.5.2 Fetch Pump where pond ID is user selection

```
select *
from pump
where pumpID=?
```

Reference: fetchPump function in pondSimDatabase, lines 20 to 27

5.5.3 Fetch Filter where pond ID is user selection

```
select *
from filter
where filterID=?
```

Reference: fetchFilter function in pondSimDatabase, lines 29 to 36

5.5.4 Fetch all fish from specified pond

```
select *
from fish
where pondID=?
```

Reference: fetchFish function in pondSimDatabase, lines 38 to 45

5.5.5 Fetch all Plants from a specified Pond

```
select *
from plant
where pondID=?
```

Reference: fetchPlants function in pondSimDatabase, lines 47 to 54

5.5.6 Fetch everything from the PlantType Table

```
select *
from plantType
```

Reference: fetchPlantType function in pondSimDatabase, lines 65 to 70

5.5.7 Fetch everything from the FishType table

```
select *
from fishType
```

Reference: fetchFishType function in pondSimDatabase, lines 72 to 77

5.5.8 Fetch all Growth Rates for the Different Fish Types

```
select growthRate
from fishType
```

Reference: fetchGrowthRate function in pondSimDatabase, lines 79 to 84

5.5.9 Deleting a Specified Pond

```
delete
from pond
where pondID=?
```

Reference: deletePond function in pondSimDatabase, lines 102 to 108

5.5.10 Fetching all pond IDs

```
select pondID
from pond
```

Reference: fetchPondID function in pondSimDatabase, lines 110 to 115

5.5.11 Deleting a Specified Fish from Fish Database

```
delete
from fish
where fishID=?
```

Reference: deleteFish function in pondSimDatabase, lines 117 to 123

5.5.12 Deleting all Fish from Specified Pond

```
delete
from fish
where pondID=?
```

Reference: deleteFishFromPond function in pondSimDatabase, lines 125 to 131

5.5.13 Deleting all plants from Specified Pond

```
delete
from plant
where pondID=?
```

Reference: deletePlantFromPond function in pondSimDatabase, lines 133 to 139

5.5.14 Deleting Specified Plant

```
delete
from plant
where plantID=?
```

Reference: deletePlant function in pondSimDatabase, lines 141 to 147

5.5.15 Delete Pump

```
delete
from pump
where pumpID=?
```

Reference: deletePump function in pondSimDatabase, lines 149 to 155

5.5.16 Delete Filter

```
delete
from filter
where filterID=?
```

Reference: deleteFilter function in pondSimDatabase, lines 157 to 163

5.5.17 Save Fish to Database

```
sql = """insert into fish(fishName, hunger, status, daysAlive, growth,
fishType, pondID) values
                (
'{0}','{1}','{2}','{3}','{4}','{5}','{6}')""".format(self._fishName,
self._hunger, self._status, self._daysAlive, self._growth, self._fishType,
pondID)
```

I included some of the surrounding python code for this query to make it easier to undersand.

Reference: saveFish method in the fish class in the fish module, lines 78 to 84

5.5.18 Save Plants to Database

```
sql = """insert into plant(status, lightNeed, daysAlive, growth, plantType,
PondID) values
                (
'{0}','{1}','{2}','{3}','{4}','{5}')""".format(self._status,
self._lightNeed, self._daysAlive, self._growth, self._plantType, pondID)
```

Reference: savePlant method in the plant class in the plant module, lines 46 to 52

5.5.19 Save Pond to Database

```
sql = """insert into pond(pondDepth, pondLength, pondWidth, waterTemp,
waterLevel, pumpID, filterID, days) values
                (
'{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}')""".format(self._pondDepth,
self._pondLength, self._pondWidth, self._waterTemp, self._waterLevel,
self._pump, self._filter, self._days)
```

Reference: savePond method in the pond class in the pond module, lines 169 to 186

5.5.19 Update Pond

```
update pond
set pondDepth=?, pondLength=?, pondWidth=?, waterTemp=?, waterLevel=?,
days=?, pumpID=?, filterID=?
where pondID=?"""
```

Reference: updatePond method in the pond class in the pond module, lines 188 to 206

5.5.20 Update Filter

```
update filter
set filterStatus=?
where filterID=?
```

Reference: updateFilter method in the filter class in the pond module, lines 231 to 237

5.5.21 Update Pump

```
update pump
set pumpStatus=?
where pumpID=?
```

Reference: updatePump method in the pump class in the pond module, lines 262 to 268

5.5.22 Update Fish

```
update fish
set hunger=?, daysAlive=?
where fishID=?"""
```

Reference: updateFish function in the incrementDay function in the pond module, lines 434 to 439

5.5.23 Update Plant

```
update plant
set daysAlive=?, growth=?
where plantID=?"""
```

Reference: updatePlant function in the incrementDay function in the pond module, lines 442 to 446

6. Testing

6.1 Summary of Results

My testing proved my program to be reliable and rebost as results were as expected and the program did not crash during testing (see 3.1 Actual Results, page 75).

The weaknesses of my testing program are that I neglected to test several features of my GUI, such as the radio buttons. I also did not test specific queries, although a lot of my tests relied on them working so I believe most or all to be correct. I failed to test if trying to perform certain functions while a pond is not open would cause the program to crash – this would obviously affect the robustness of my program

The main strength of my testing was when I tested the GUI features of my program – it proved that my program could be navigated correctly and easily through the menu bar and buttons, and that the graphical display is correct. The testing revealed that all of my buttons and menu options were linked to their correct places, and that the graphical display displayed all of the fish and plants correctly. My testing also allowed me to identify the problem discussed below.

6.2 Known Issues

Test 4.1 - The fish status did not change after feeding

I intended the status of a fish to change to dead from alive once the fish reached over 20 hunger, however this only happens when the day is incremented and not when the fish are fed (or not fed) even though the day is incremented as part of that function. I'm not sure how to fix this error other than checking the fish's hunger level in the manual feeding function (10.5 Pond Module, page 167 – lines 648 to 692) as well as the increment day function. The fish's status might not be saving properly.

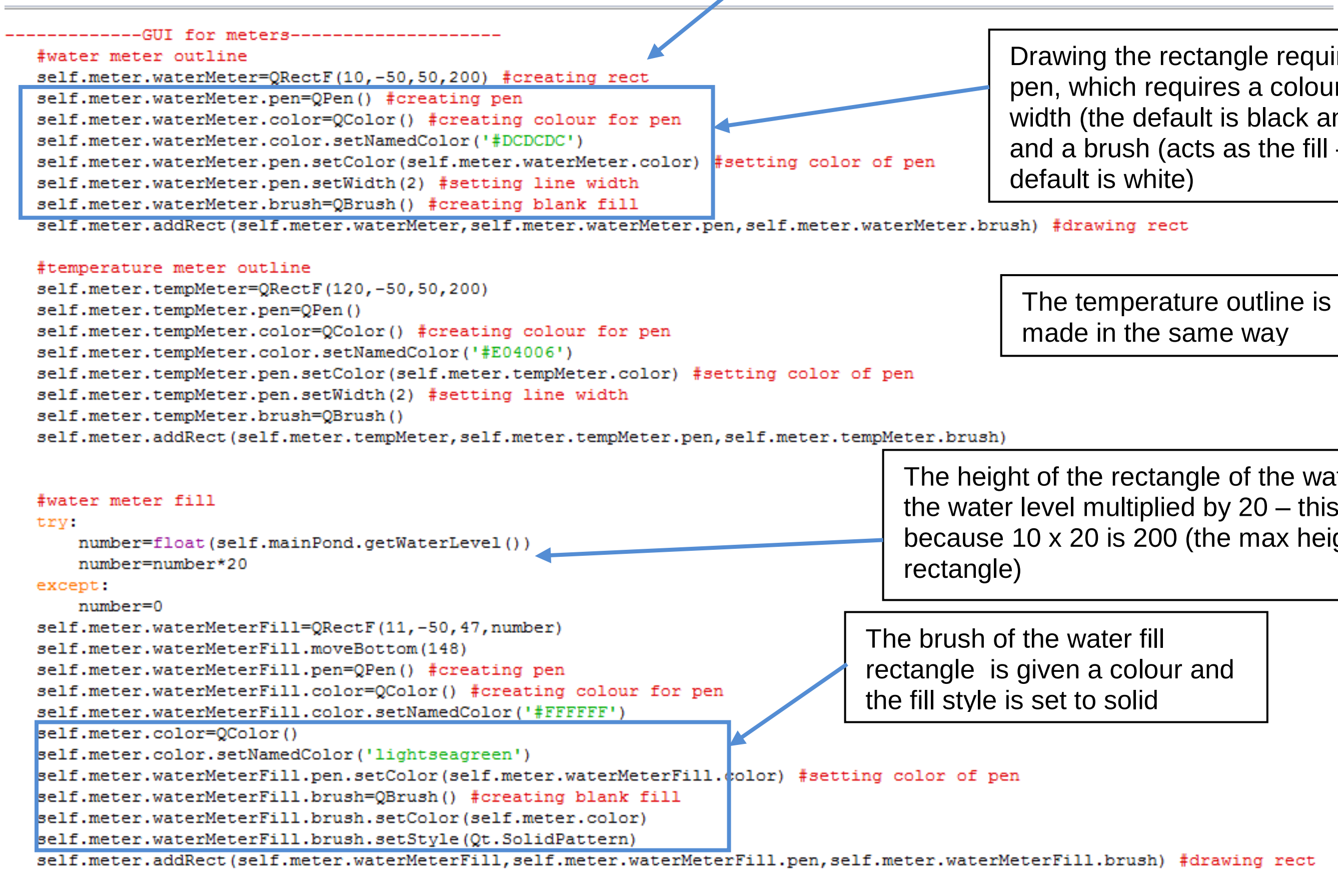
7. Code Explanations

7.1 Difficult Sections

The following section is lines 1099 to 1208 in the GUI code section, which can be found in the section appendix. The purpose of this section of code is to draw the water and temperature meters and display them in the graphic display.

Defining the size of the rect for the water meter outline

Drawing the rectangle requires a pen, which requires a colour and a width (the default is black and 1) and a brush (acts as the fill – default is white)



```
-------------GUI for meters--------------------
   #water meter outline
   self.meter.waterMeter=QRectF(10,-50,50,200) #creating rect
   self.meter.waterMeter.pen=QPen() #creating pen
   self.meter.waterMeter.color=QColor() #creating colour for pen
   self.meter.waterMeter.color.setNamedColor('#DCDCDC')
   self.meter.waterMeter.pen.setColor(self.meter.waterMeter.color) #setting color of pen
   self.meter.waterMeter.pen.setWidth(2) #setting line width
   self.meter.waterMeter.brush=QBrush() #creating blank fill
   self.meter.addRect(self.meter.waterMeter,self.meter.waterMeter.pen,self.meter.waterMeter.brush) #drawing rect

   #temperature meter outline
   self.meter.tempMeter=QRectF(120,-50,50,200)
   self.meter.tempMeter.pen=QPen()
   self.meter.tempMeter.color=QColor() #creating colour for pen
   self.meter.tempMeter.color.setNamedColor('#E04006')
   self.meter.tempMeter.pen.setColor(self.meter.tempMeter.color) #setting color of pen
   self.meter.tempMeter.pen.setWidth(2) #setting line width
   self.meter.tempMeter.brush=QBrush()
   self.meter.addRect(self.meter.tempMeter,self.meter.tempMeter.pen,self.meter.tempMeter.brush)


   #water meter fill
   try:
       number=float(self.mainPond.getWaterLevel())
       number=number*20
   except:
       number=0
   self.meter.waterMeterFill=QRectF(11,-50,47,number)
   self.meter.waterMeterFill.moveBottom(148)
   self.meter.waterMeterFill.pen=QPen() #creating pen
   self.meter.waterMeterFill.color=QColor() #creating colour for pen
   self.meter.waterMeterFill.color.setNamedColor('#FFFFFF')
   self.meter.color=QColor()
   self.meter.color.setNamedColor('lightseagreen')
   self.meter.waterMeterFill.pen.setColor(self.meter.waterMeterFill.color) #setting color of pen
   self.meter.waterMeterFill.brush=QBrush() #creating blank fill
   self.meter.waterMeterFill.brush.setColor(self.meter.color)
   self.meter.waterMeterFill.brush.setStyle(Qt.SolidPattern)
   self.meter.addRect(self.meter.waterMeterFill,self.meter.waterMeterFill.pen,self.meter.waterMeterFill.brush) #drawing rect
```

The temperature outline is made in the same way

The height of the rectangle of the water fill is the water level multiplied by 20 – this is because 10 x 20 is 200 (the max height of the rectangle)

The brush of the water fill rectangle is given a colour and the fill style is set to solid

This determines the height of the rectangle. If the water temperature is greater than 0, it means the temperature is positive. The number is times by two to fit in the box properly

```
#water temperature fill
try:
    number=float(self.mainPond.getWaterTemp())
    if number>=0:
        number=number*2
    else:
        number=0
except:
    number=0
self.meter.tempMeterFill=QRectF(122,-50,45,number)
self.meter.tempMeterFill.moveBottom(48)
self.meter.tempMeterFill.pen=QPen()#creating pen
color=QColor() #creating colour for pen
color.setNamedColor('#FFFFFF')
self.meter.tempMeterFill.color=QColor()
self.meter.tempMeterFill.color.setNamedColor('red')
self.meter.tempMeterFill.pen.setColor(self.meter.tempMeterFill.color) #setting color of pen
self.meter.tempMeterFill.brush=QBrush() #creating blank fill
self.meter.tempMeterFill.brush.setColor(self.meter.tempMeterFill.color)
self.meter.tempMeterFill.brush.setStyle(Qt.SolidPattern)
self.meter.addRect(self.meter.tempMeterFill,self.meter.tempMeterFill.pen,self.meter.tempMeterFill.brush) #drawing rect
```

The rectangle is drawn in the middle of the outline rectangle

This determines the height of the rectangle. If the water temperature is less than 0, the height is minus the water temperature multiplied by 2, otherwise the height is 100 (the full height of the box)

```
#water temperature minus fill
try:
    number=float(self.mainPond.getWaterTemp())
    if number<0:
        number=-number*2
    else:
        number=100
except:
    number=0
self.meter.tempMeterFill=QRectF(122,-50,45,number)
self.meter.tempMeterFill.moveBottom(147)
self.meter.tempMeterFill.pen=QPen() #creating pen
color=QColor() #creating colour for pen
color.setNamedColor('#FFFFFF')
self.meter.tempMeterFill.color=QColor()
self.meter.tempMeterFill.color.setNamedColor('blue')
self.meter.tempMeterFill.pen.setColor(self.meter.tempMeterFill.color) #setting color of pen
self.meter.tempMeterFill.brush=QBrush() #creating blank fill
self.meter.tempMeterFill.brush.setColor(self.meter.tempMeterFill.color)
self.meter.tempMeterFill.brush.setStyle(Qt.SolidPattern)
self.meter.addRect(self.meter.tempMeterFill,self.meter.tempMeterFill.pen,self.meter.tempMeterFill.brush) #drawing rect

self.widgetM=QWidget() #creating widget
try:
    level=str(self.mainPond.getWaterLevel())
except:
    level=""
try:
    temp=str(self.mainPond.getWaterTemp())
except:
    temp=""
self.waterLabel=QLabel("Water level: "+level)#making labels
self.tempLabel=QLabel("Temperature: "+temp)
self.zeroLabel=QLabel("  0")
self.zeroLabel.setFixedHeight(8)
self.fiftyLabel=QLabel("  50")
self.fiftyLabel.setFixedHeight(185)
```

7.2. Self-created Algorithms

7.2.1 Automatically Grow Pond

Reference: 10.5 Pond Module 167, line 695 to 732.

The following function looks up what plants and fish are currently in the pond and grows them with the exact required food for 7 days.

```
def automaticGrow(db,cursor,pond):
    #feeds fish the required food amount for 7 days.
    def updateFish(tup):
        sql = """update fish
                set hunger=?, growth=?, status=?
                where fishID=?"""
        cursor.execute(sql,tup)
        db.commit()

    fishList=pondSimDatabase.fetchFishInfo(db,cursor,pond.getPondID())
    growthList=pondSimDatabase.fetchGrowthRate(db,cursor)
    for count in range(0,7):
        for count in range(0,len(fishList)):
            if fishList[count][3]!="Dead":
                if fishList[count][6]=="Goldfish":
                    growth=growthList[0][0]+fishList[count][5]
                elif fishList[count][6]=="Tench":
                    growth=growthList[1][0]+fishList[count][5]
                elif fishList[count][6]=="Orfe":
                    growth=growthList[2][0]+fishList[count][5]
                elif fishList[count][6]=="Koi":
                    growth=growthList[3][0]+fishList[count][5]
                elif fishList[count][6]=="Rudd":
                    growth=growthList[4][0]+fishList[count][5]
                elif fishList[count][6]=="Shark":
                    growth=growthList[5][0]+fishList[count][5]
            else:
                growth=fishList[count][5]
            if fishList[count][2]>20:
                status="Dead"
                growth=fishList[count][5]
            else:
                status=fishList[count][3]
            hunger=0
            fishID=fishList[count][0]
            tup=(hunger,growth,status,fishID)
            updateFish(tup)
        incrementDay(pond,db,cursor)
```

This function will save the fish information to the database

Fetches all the fish in the pond from the database

Fetches all the growth rates of the fish

Goes through each fish at a time and adds their growth rate to their growth level, if they are not dead. If they are dead, their growth level stays the same. If the fish is over 20 hunger, their status is changed to dead. Hunger is set to 0, and "updateFish" is called to save the changed data to the database. The day is incremented. This whole loop repeats 7 time to simulate the 7 days.

7.2.2 Manually Grow Pond

Reference: 10.5 Pond Module 167, line 648 to 692

The following function receives an input from the user for how food they want to add to the pond, if the fish are completely fed, they will grow and the pond will increment by one day.

“amount” is the amount of food the user wants to add to the pond – it’s already been decided

```
def manualGrow(amount,pond,db,cursor):
    def updateFish(tup):
        sql = """update fish
                set hunger=?, growth=?, status=?
                where fishID=?"""
        cursor.execute(sql,tup)
        db.commit()
```

This function will save the fish information to the database

Fetches a list of all the fish in the pond

```
    hunger=pondSimDatabase.fetchFishInfo(db,cursor,pond.getPondID())
    for count in range(0,len(hunger)):
        amount2=amount
        amount=amount-hunger[count][2]
        hunger2=hunger[count][2]-amount2
        if hunger2<0:
            hunger2=0
```

The fish’s hunger level is minused from the food amount to get the amount of food left. The fish’s hunger has the food amount minused from it to get the new hunger level. If the fish’s hunger is less than 0, it’s changed to 0 (you can’t be negative hungry). The loop repeats for all the fish in the list.

```
        growthList=pondSimDatabase.fetchGrowthRate(db,cursor)
        if hunger2==0 and hunger[count][3]!="Dead":
            if hunger[count][6]=="Goldfish":
                growth=growthList[0][0]+hunger[count][5]
            elif hunger[count][6]=="Tench":
                growth=growthList[1][0]+hunger[count][5]
            elif hunger[count][6]=="Orfe":
                growth=growthList[2][0]+hunger[count][5]
            elif hunger[count][6]=="Koi":
                growth=growthList[3][0]+hunger[count][5]
            elif hunger[count][6]=="Rudd":
                growth=growthList[4][0]+hunger[count][5]
            elif hunger[count][6]=="Shark":
                growth=growthList[5][0]+hunger[count][5]
```

If the fish has been completely fed and is still alive, its growth rate will be added to its current growth level

```
        else:
            growth=hunger[count][5]
        fishID=hunger[count][0]
```

If the fish is still hungry or is dead, its growth rate stays the same

```
        if hunger[count][2]>20:
            status="Dead"
            growth=hunger[count][5]
        else:
            status=hunger[count][3]
```

If fish’s hunger is greater than 20, its staus is changed to dead, else it stays the same.

```
        #hunger2=0
        fishID=hunger[count][0]
        tup=(hunger2,growth,status,fishID)
        updateFish(tup)
    incrementDay(pond,db,cursor)
```

The fish is saved

The day is incremented

7.2.3 Increment Day

Reference: 10.5 Pond Module 167, line 433 to 520

The following function increments the number of days the pond has been around and all the plants and fish in it.

```
def incrementDay(mainPond,db,cursor):
    def updateFish(tup):
        sql = """update fish
                set hunger=?, daysAlive=?
                where fishID=?"""
        cursor.execute(sql,tup)
        db.commit()

    def updatePlants(tup):
        sql="""update plant
            set daysAlive=?, growth=?
            where plantID=?"""
        cursor.execute(sql,tup)
        db.commit()

    fishList=pondSimDatabase.fetchFishInfo(db,cursor,mainPond.getPondID())
    plantList=pondSimDatabase.fetchPlantInfo(db,cursor,mainPond.getPondID())
    plantType=pondSimDatabase.fetchPlantType(db,cursor)
    foodNeed=pondSimDatabase.fetchFoodNeed(db,cursor)

    for count in range(0,len(fishList)):
        daysAlive=fishList[count][4]+1
        if fishList[count][6]=="Goldfish":
            hunger=fishList[count][2]+foodNeed[0][1]
        elif fishList[count][6]=="Tench":
            hunger=fishList[count][2]+foodNeed[1][1]
        elif fishList[count][6]=="Orfe":
            hunger=fishList[count][2]+foodNeed[2][1]
        elif fishList[count][6]=="Koi":
            hunger=fishList[count][2]+foodNeed[3][1]
        elif fishList[count][6]=="Rudd":
            hunger=fishList[count][2]+foodNeed[4][1]
        elif fishList[count][6]=="Shark":
            hunger=fishList[count][2]+foodNeed[5][1]
        if fishList[count][3]=="Dead":
            hunger=0
            daysAlive=0
        elif fishList[count][3]=="good":
            daysAlive=fishList[count][4]+1
        tup=(hunger,daysAlive,fishList[count][0])
        updateFish(tup)
```

Updates the fish in the database

Updates the plants in the database

Fetching list of all fish, list of all plants, list of the plant type information, list of the fish type informaion

Goes through each fish in the fish list and adds the food need of that fish type to the fish's hunger level to calculate the new hunger level.

If the fish is dead, hunger is set to 0 and days alive is set to 0, else the days alive is incremeneted by 1.

The fish is updated in the database

Continues next page

```
for count in range(0,len(plantList)):
        daysAlive=plantList[count][4]+1
        if plantList[count][1]=="Rooted Floater":
            growth=plantList[count][5]+plantType[0][1]
        elif plantList[count][1]=="True Floater":
            growth=plantList[count][5]+plantType[1][1]
        elif plantList[count][1]=="Emergent":
            growth=plantList[count][5]+plantType[2][1]
        elif plantList[count][1]=="submersed":
            growth=plantList[count][5]+plantType[3][1]
        if plantList[count][2]=="Dead":
            daysAlive=0
        else:
            daysAlive=plantList[count][4]+1
        tup=(daysAlive,plantList[count][5]+1,plantList[count][0])
        updatePlants(tup)
```

Goes through each plant in the plant list and adds the plant's growthrate to the growth of the plant.

If the plant is dead, the number of days alive is set to 0, else it's incremented by 1.

The plant is updated in the database

```
    randomNo=random.randint(0,20)
    if randomNo==1:
       try:
           pump=pondSimDatabase.fetchPump(db,cursor,mainPond.getPump())
#fetching pump data
           pump=Pump(pump[0][1], pump[0][0]) #creating pump object from
pump data
           pumpStatus=pump.getPumpStatus()
           pump.setPumpStatus("Broken")
           pump.updatePump(db,cursor)
           mainPond.setPump(pump.getPumpID())
       except:
            pass
```

Generates a random number between 1 and 20. If number equals 1, the pump's status is changed to "Broken" and the change is saved to the database.

```
    randomNo=random.randint(0,20)
    if randomNo==1:
        try:

filter=pondSimDatabase.fetchFilter(db,cursor,self.mainPond.getFilter())
#fetching pump data
            filter=Filter(filter[0][1], filter[0][0]) #creating pump object
from pump data
            filterStatus=filter.getFilterStatus()
            filter.setFilterStatus("Broken")
            filter.updateFilter(db,cursor)
            mainPond.setFilter(filter.getFilterID())
        except:
            pass


    mainPond.setDays(int(mainPond.getDays())+1)
    mainPond.updatePond(db,cursor)
```

Generates a random number between 1 and 20. If number equals 1, the filter's status is changed to "Broken" and the change is saved to the database.

7.2.4 Add Plants

Reference: 10.2 GUI, page 139 – lines 1023 to 1063

The following function fetches the plants stored in the database and adds an image of them (depending on their type and status) to the pond in a randomly generated position.

Fetchs a list of all the plants in the pond from the database

Section will repeat for all plants in the list

If plant is a rooted floater and status is good, alive rooted floater graphic is made. If status is dead, dead graphic is added

If plant is a true floater and status is good, alive true floater graphic is made. If status is dead, dead graphic is added

If plant is an emergent and status is good, alive emergent plant graphic is made. If status is dead, dead graphic is added

If plant is an submersed and status is good, alive submersed plant graphic is made. If status is dead, dead graphic is added

Created graphic is added to the graphics scene at a randomly generated position.

```
        def addPlants():
plantResult=pondSimDatabase.fetchPlants(db,cursor,self.mainPond.getPondID()
)
            for count in range(0,len(plantResult)):
                if plantResult[count][1]=="Rooted Floater":
                    if plantResult[count][2]=="good":
                        self.graphics.plant=QPixmap("rootedFloater.png")
                    elif plantResult[count][2]=="Dead":
                        self.graphics.plant=QPixmap("rootedFloater.png")
                elif plantResult[count][1]=="True Floater":
                    if plantResult[count][2]=="good":
                        self.graphics.plant=QPixmap("trueFloater.png")
                    elif plantResult[count][2]=="Dead":
                        self.graphics.plant=QPixmap("trueFloater.png")
                elif plantResult[count][1]=="Emergent":
                    if plantResult[count][2]=="good":
                        self.graphics.plant=QPixmap("rootedFloater.png")
                    elif plantResult[count][2]=="Dead":
                        self.graphics.plant=QPixmap("rootedFloater.png")
                elif plantResult[count][1]=="Submersed":
                    if plantResult[count][2]=="good":
                        self.graphics.plant=QPixmap("rootedFloater.png")
                    elif plantResult[count][2]=="Dead":
                        self.graphics.plant=QPixmap("rootedFloater.png")
self.graphics.plantItem=self.graphics.addPixmap(self.graphics.plant)
                self.graphics.plantItem.setPos(random.randrange(1,549),0)
                self.graphics.plantItem.z=0
```

7.2.5 Add Fish

Reference: 10.2 GUI, page 139 – lines 976 to 1031

The following function fetches the fish stored in the database and adds an image of them (depending on the type and status) to the pond in a randomly generated position.

Fetchs a list of all the fish in the pond from the database

```
        def addFish():

fishResults=pondSimDatabase.fetchFish(db,cursor,self.mainPond.getPondID())
            for count in range(0,len(fishResults)):
                if fishResults[count][6]=="Goldfish":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
```

If fish is a goldfish and status is good, alive goldfish graphic is made. If status is dead, dead graphic is added

Adds fish graphic to graphics scene at a random postion

```
self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)

self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,39
9)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2
                elif fishResults[count][6]=="Tench":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
```

If fish is a tench and status is good, alive tench graphic is made. If status is dead, dead graphic is added

```
self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)

self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,39
9)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2
                elif fishResults[count][6]=="Orfe":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
```

If fish is an orfe and status is good, alive orfe graphic is made. If status is dead, dead graphic is added

```
self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)

self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,39
9)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2
                elif fishResults[count][6]=="Koi":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
```

If fish is a koi and status is good, alive koi graphic is made. If status is dead, dead graphic is added

```
self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)

self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,39
9)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2
```

If fish is a rudd and status is good, alive rudd graphic is made. If status is dead, dead graphic is added

```
elif fishResults[count][6]=="Rudd":
    if fishResults[count][3]=="good":
        self.graphics.fish=QPixmap("fish.png")
    elif fishResults[count][3]=="Dead":
        self.graphics.fish=QPixmap("dead.png")
```

```
self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)

self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,39
9)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2
```

If fish is a shark and status is good, alive shark graphic is made. If status is dead, dead graphic is added

```
elif fishResults[count][6]=="Shark":
    if fishResults[count][3]=="good":
        self.graphics.fish=QPixmap("shark.png")
    elif fishResults[count][3]=="Dead":
        self.graphics.fish=QPixmap("sharkDead.png")
```

```
self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)
                    self.graphics.fishItem.setPos(random.randrange(-
10,10),random.randrange(-20,20)) #adds the fish graphic at a random
location
                    self.graphics.fishItem.z=2
```

8. Settings

In order for my program to run correctly, Python 3.2 must be installed on the machine as well the modules SQLite3and PyQt4

9. Acknowledgements

I referenced this page on ZetCode when drawing my graphics http:://zetcode.com/tutorials/pyqt4/drawing/

The code I looked at is as follows:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
ZetCode PyQt4 tutorial

This example draws three rectangles in three
different colors.

author: Jan Bodnar
website: zetcode.com
last edited: September 2011
"""

import sys
from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):
    def __init__(self):
        super(Example, self).__init__()

        self.initUI()

    def initUI(self):

        self.setGeometry(300, 300, 350, 100)
        self.setWindowTitle('Colors')
        self.show()

    def paintEvent(self, e):

        qp = QtGui.QPainter()
        qp.begin(self)
        self.drawRectangles(qp)
        qp.end()

    def drawRectangles(self, qp):

        color = QtGui.QColor(0, 0, 0)
        color.setNamedColor('#d4d4d4')
        qp.setPen(color)

        qp.setBrush(QtGui.QColor(200, 0, 0))
        qp.drawRect(10, 15, 90, 60)

        qp.setBrush(QtGui.QColor(255, 80, 0, 160))
        qp.drawRect(130, 15, 90, 60)

        qp.setBrush(QtGui.QColor(25, 0, 90, 200))
        qp.drawRect(250, 15, 90, 60)


def main():

    app = QtGui.QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())


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

10. Code Listing Appendix

10.1 databaseCreation Module

```
import sqlite3
import fish
import plant

#creating databases
def createPondTable(db,cursor):
    sql = """create table pond(
            pondID integer,
            pondDepth integer,
            pondLength integer,
            pondWidth integer,
            waterTemp integer,
            waterLevel integer,
            pumpID integer,
            filterID integer,
            Days integer,
            primary key(pondID),
            foreign key(pumpID) references pump(pumpID) on update cascade on delete restrict,
            foreign key(filterID) references filter(filterID) on update cascade on delete restrict)"""
    cursor.execute(sql)

def createFishTypeTable(db,cursor):
    sql = """create table fishType(
            fishType text,
            foodNeed integer,
            growthRate integer,
            fishSize integer,
            primary key(fishType))"""
    cursor.execute(sql)

def createFishTable(db,cursor):
    sql = """create table fish(
            fishID integer,
            fishName text,
            hunger integer,
            status text,
            daysAlive integer,
            growth integer,
            fishType text,
            pondID integer,
            primary key(fishID),
            foreign key(fishType) references fishType(fishType),
            foreign key(pondID) references pond(pondID))"""
    cursor.execute(sql)

def createPlantTypeTable(db,cursor):
    sql = """create table plantType(
            plantType text,
            lightRequirement integer,
            growthRate integer,
            primary key(plantType))"""
    cursor.execute(sql)

def createPlantTable(db,cursor):
    sql = """create table plant(
            plantID integer,
            plantType text,
            status text,
            lightNeed integer,
            daysAlive integer,
            growth integer,
            PondID integer,
            primary key(plantID),
            foreign key(plantType) references plantType(plantType),
            foreign key(pondID) references pondID(pondID))"""
    cursor.execute(sql)

def createPumpTable(db,cursor):
    sql = """create table pump(
            pumpID integer,
            pumpStatus text,
            primary key (pumpID))"""
    cursor.execute(sql)

def createFilterTable(db,cursor):
    sql = """create table filter(
            filterID integer,
            filterStatus text,
            primary key (filterID))"""
    cursor.execute(sql)

def createDatabases():
    db=sqlite3.connect('pond.db')
    cursor=db.cursor()
    try:
        createPumpTable(db,cursor)
    except:
        pass

    try:
        createFilterTable(db,cursor)
    except:
        pass

    try:
        createFishTypeTable(db,cursor)
    except:
        pass

    try:
        createPlantTypeTable(db,cursor)
    except:
        pass

    try:
        createPondTable(db,cursor)
    except:
        pass

    try:
        createFishTable(db,cursor)
    except:
        pass

    try:
        createPlantTable(db,cursor)
    except:
        pass
    fish.saveFishTypes(db,cursor)
    plant.savePlantTypes(db,cursor)
    cursor.close()
```

10.2 GUI

```
from PyQt4.QtGui import *
from PyQt4.QtCore import *
import sys
import pond
import sqlite3
import pondSimDatabase
import databaseCreation
import fish
import plant
import random

class MainWindow(QMainWindow): #main window class inherits from QMainWindow
    def __init__(self): #constructor
        super(MainWindow,self).__init__() #call parent constructor
        self.createUI() #call creatUI method to create user interface for the main window

    def createUI(self):
        def savePond():
            try:
                self.mainPond.updatePond(db,cursor)
            except:
                self.errorWindow=QWidget()
                self.errorWindow.setWindowTitle("Error")
                self.errorWindow.setFixedWidth(250)
                self.errorWindow.setFixedHeight(150)
                self.errorWindow.show()
                self.errorWindow.layout = QVBoxLayout()
                self.errorWindow.setLayout(self.errorWindow.layout)
                self.errorWindow.label=QLabel("You must open a pond before it can be saved")
                self.errorWindow.label.setWordWrap(True)
                self.errorWindow.widget=QWidget()
                self.errorWindow.widgetLayout=QGridLayout()
                self.errorWindow.widget.setLayout(self.errorWindow.widgetLayout)
                self.errorWindow.okay=QPushButton("OK")
                self.errorWindow.okay.setFixedWidth(50)
                self.errorWindow.okay.clicked.connect(self.errorWindow.close)
                blank=QWidget()
                blank.setFixedWidth(100)
                blankTwo=QWidget()
                blankTwo.setFixedWidth(100)
                self.errorWindow.widgetLayout.addWidget(blank,0,2)
                self.errorWindow.widgetLayout.addWidget(blankTwo,0,0)
                self.errorWindow.widgetLayout.addWidget(self.errorWindow.okay,0,1)
                self.errorWindow.layout.addWidget(self.errorWindow.label)
                self.errorWindow.layout.addWidget(self.errorWindow.widget)

        def openWindow():
            #when pond is opened
            def onOpen():
                self.window.close() #hides window
                self.selected=self.window.tree.currentItem()
                pondID=self.selected.data(0,0)
                depth=self.selected.data(1,0)
                length=self.selected.data(2,0)
                width=self.selected.data(3,0)
                temp=self.selected.data(4,0)
                level=self.selected.data(5,0)
                pump=self.selected.data(6,0)
                Filter=self.selected.data(7,0)
                days=self.selected.data(8,0)
                self.mainPond=pond.Pond(pondID=pondID,depth=depth,width=width,temp=temp,waterLevel=level,pump=pump,filter=Filter,days=days, length=length)
                self.createUI()

            def tree():
                self.window.tree.setHeaderItem(self.window.header) #setting header
                results=pondSimDatabase.getPonds(db,cursor)
                for count in range(0,len(results)):
                    self.window.treeList=QTreeWidgetItem()#list items
                    self.window.treeList.setText(0,str(results[count][0]))
                    self.window.treeList.setText(1,str(results[count][1]))
                    self.window.treeList.setText(2,str(results[count][2]))
                    self.window.treeList.setText(3,str(results[count][3]))
                    self.window.treeList.setText(4,str(results[count][4]))
                    self.window.treeList.setText(5,str(results[count][5]))
                    self.window.treeList.setText(6,str(results[count][6]))
                    self.window.treeList.setText(7,str(results[count][7]))
                    self.window.treeList.setText(8,str(results[count][8]))
                    self.window.tree.addTopLevelItem(self.window.treeList)

            self.window=QWidget()
            self.window.setWindowTitle("Pond Simulator 3000 - Open Pond")
            self.window.setFixedWidth(600)
            self.window.setFixedHeight(400)
            self.window.show()
            self.window.layout = QVBoxLayout()
            self.window.setLayout(self.window.layout)

            self.window.tree=QTreeWidget()
            self.window.tree.setColumnCount(9)
            self.window.tree.setColumnWidth(0,80)

            self.window.header=QTreeWidgetItem() #headers
            self.window.header.setText(0,'PondID')
            self.window.header.setText(1,'Pond Depth')
            self.window.header.setText(2,'Pond Length')
            self.window.header.setText(3,'Pond Width')
            self.window.header.setText(4,'Water Temp.')
            self.window.header.setText(5,'Water Level')
            self.window.header.setText(6,'PumpID')
            self.window.header.setText(7,'FilterID')
            self.window.header.setText(8,'Days')
            tree()

            self.window.button=QPushButton("Cancel") #making button
            self.window.button.setFixedWidth(100) #changing button size
            self.window.button.layout=QHBoxLayout() #creating layout for button
            self.window.buttonWidget=QWidget() #creating blank widget
            self.window.blank=QWidget()
            self.window.buttonWidget.setLayout(self.window.button.layout) #setting layout to blank widget
            self.window.button.layout.addWidget(self.window.blank) #adding button layout
            self.window.button.layout.addWidget(self.window.button) #adding button layout

            self.window.button.clicked.connect(self.window.close) #when button is clicked, window is hidden
            self.window.layout.addWidget(self.window.tree) #adding tree list to layout
            self.window.layout.addWidget(self.window.buttonWidget) #adding button to layout
            self.window.tree.itemDoubleClicked.connect(onOpen)#when item double clicked

        def newPondWindow():
            self.window=QWidget()
            self.window.setWindowTitle("Pond Simulator 3000 - New Pond")
            self.window.setFixedWidth(600)
            self.window.setFixedHeight(400)
            self.window.show()
            self.window.layout = QVBoxLayout()
            self.window.setLayout(self.window.layout)

            #box one
            self.window.mainWidget=QFrame()
            self.window.mainWidget.setLineWidth(1)
            self.window.mainWidget.setFrameShape(QFrame.StyledPanel)
            self.window.mainWidgetLayout=QVBoxLayout()
            self.window.mainWidget.setLayout(self.window.mainWidgetLayout)

            self.window.labelWidget=QWidget()
            self.window.labelWidgetLayout=QHBoxLayout()
            self.window.labelWidget.setLayout(self.window.labelWidgetLayout)
            self.window.labelWidgetLayout.addWidget(QLabel("The pond can be a maxium of 10 x 10 x 10 metres"))

            self.window.editWidget=QWidget()
            self.window.editWidgetLayout=QGridLayout()
            self.window.editWidget.setLayout(self.window.editWidgetLayout)
            self.window.editWidgetLayout.addWidget(QLabel("Depth:"),0,0)
            self.window.depthLine=QSpinBox()
            self.window.depthLine.setMaximum (10)
            self.window.depthLine.setMinimum(0.5)
            self.window.editWidgetLayout.addWidget(self.window.depthLine,0,1)
            self.window.editWidgetLayout.addWidget(QLabel("Width:"),0,2)
            self.window.widthLine=QSpinBox()
            self.window.widthLine.setMaximum (10)
            self.window.widthLine.setMinimum(0.5)
            self.window.editWidgetLayout.addWidget(self.window.widthLine,0,3)
            self.window.editWidgetLayout.addWidget(QLabel("Length:"),0,4)
            self.window.lengthLine=QSpinBox()
            self.window.lengthLine.setMaximum (10)
            self.window.lengthLine.setMinimum(0.5)
            self.window.editWidgetLayout.addWidget(self.window.lengthLine,0,5)

            self.window.mainWidgetLayout.addWidget(self.window.labelWidget)
            self.window.mainWidgetLayout.addWidget(self.window.editWidget)
            self.window.layout.addWidget(self.window.mainWidget)

            #box two
            self.window.mainWidget=QFrame()
            self.window.mainWidget.setLineWidth(1)
            self.window.mainWidget.setFrameShape(QFrame.StyledPanel)
            self.window.mainWidgetLayout=QVBoxLayout()
            self.window.mainWidget.setLayout(self.window.mainWidgetLayout)

            self.window.labelWidget=QWidget()
            self.window.labelWidgetLayout=QHBoxLayout()
            self.window.labelWidget.setLayout(self.window.labelWidgetLayout)
            self.window.labelWidgetLayout.addWidget(QLabel("The water temperature must be between 27 and +50C."))

            self.window.editWidget=QWidget()
            self.window.editWidgetLayout=QGridLayout()
            self.window.editWidget.setLayout(self.window.editWidgetLayout)
            self.window.editWidgetLayout.addWidget(QLabel("Temperature:"),0,0)
            self.window.tempLine=QSpinBox()
            self.window.tempLine.setMaximum (50)
            self.window.tempLine.setMinimum(-27)
            self.window.editWidgetLayout.addWidget(self.window.tempLine,0,1)
            self.window.blank=QWidget()
            self.window.blank.setFixedWidth(2000)
            self.window.editWidgetLayout.addWidget(self.window.blank,0,2)

            self.window.mainWidgetLayout.addWidget(self.window.labelWidget)
            self.window.mainWidgetLayout.addWidget(self.window.editWidget)
            self.window.layout.addWidget(self.window.mainWidget)

            #box three
            self.window.mainWidget=QFrame()
            self.window.mainWidget.setLineWidth(1)
            self.window.mainWidget.setFrameShape(QFrame.StyledPanel)
            self.window.mainWidgetLayout=QVBoxLayout()
            self.window.mainWidget.setLayout(self.window.mainWidgetLayout)

            self.window.labelWidget=QWidget()
            self.window.labelWidgetLayout=QHBoxLayout()
            self.window.labelWidget.setLayout(self.window.labelWidgetLayout)
            self.window.labelWidgetLayout.addWidget(QLabel("Highly recommended"))

            self.window.editWidget=QWidget()
            self.window.editWidgetLayout=QGridLayout()
            self.window.editWidget.setLayout(self.window.editWidgetLayout)
            self.window.editWidgetLayout.addWidget(QLabel("Pump and filter:"),0,0)

            self.window.Radio1 = QRadioButton("Yes")
            self.window.Radio1.setChecked(True)
            self.window.Radio2 = QRadioButton("No")
            self.window.ButtonGroup = QButtonGroup()

            self.window.ButtonGroup.addButton(self.window.Radio1)
            self.window.ButtonGroup.setId(self.window.Radio1,0)
            self.window.ButtonGroup.addButton(self.window.Radio2)
            self.window.ButtonGroup.setId(self.window.Radio2,1)
            self.window.editWidgetLayout.addWidget(self.window.Radio1,0,1)
            self.window.editWidgetLayout.addWidget(self.window.Radio2,0,2)

            self.window.mainWidgetLayout.addWidget(self.window.labelWidget)
            self.window.mainWidgetLayout.addWidget(self.window.editWidget)
            self.window.layout.addWidget(self.window.mainWidget)

            #buttons
            self.createButton=QPushButton("Create")
            self.createButton.setFixedWidth(100)
            self.cancelButton=QPushButton("Cancel")
            self.cancelButton.setFixedWidth(100)
            self.blank=QWidget()
            self.blank.setFixedWidth(2000)
            self.buttonWidget=QWidget()
            self.buttonLayout=QGridLayout()
            self.buttonWidget.setLayout(self.buttonLayout)
            self.buttonLayout.addWidget(self.blank,0,0)
            self.buttonLayout.addWidget(self.createButton,0,1)
            self.buttonLayout.addWidget(self.cancelButton,0,2)
            self.window.layout.addWidget(self.buttonWidget)

            def error():
                self.errorWindow=QWidget()
                self.errorWindow.setWindowTitle("Error")
                self.errorWindow.setFixedWidth(250)
                self.errorWindow.setFixedHeight(150)
                self.errorWindow.show()
                self.errorWindow.layout = QVBoxLayout()
                self.errorWindow.setLayout(self.errorWindow.layout)
                self.errorWindow.label=QLabel("Length, width and depth values must all be greater than 0.")
                self.errorWindow.label.setWordWrap(True)
                self.errorWindow.widget=QWidget()
                self.errorWindow.widgetLayout=QGridLayout()
```

```
                self.errorWindow.widget.setLayout(self.errorWindow.widgetLayout)
                self.errorWindow.okay=QPushButton("OK")
                self.errorWindow.okay.setFixedWidth(50)
                self.errorWindow.okay.clicked.connect(self.errorWindow.close)
                blank=QWidget()
                blank.setFixedWidth(100)
                blankTwo=QWidget()
                blankTwo.setFixedWidth(100)
                self.errorWindow.widgetLayout.addWidget(blank,0,2)
                self.errorWindow.widgetLayout.addWidget(blankTwo,0,0)
                self.errorWindow.widgetLayout.addWidget(self.errorWindow.okay,0,1)
                self.errorWindow.layout.addWidget(self.errorWindow.label)
                self.errorWindow.layout.addWidget(self.errorWindow.widget)


            def getInput():
                depth=self.window.depthLine.text()
                length=self.window.lengthLine.text()
                width=self.window.widthLine.text()
                temp=self.window.tempLine.text()
                if int(depth)and int(length)and int(width)!=0:
                    result=self.window.ButtonGroup.checkedId()
                    if int(result)==0:
                        pump=pond.Pump(status="good",pumpID=None)
                        filter=pond.Filter(status="good",filterID=None)
                        filter.saveFilter(db,cursor)
                        filterID=filter.getFilterID()
                        pump.savePump(db,cursor)
                        pumpID=pump.getPumpID()
                    else:
                        pump=None
                        filter=None
                    self.mainPond=pond.Pond(pondID=None,depth=int(depth),width=int(width),temp=int(temp),waterLevel=int(depth)-0.5,pump=pumpID,filter=filterID,days=0, length=length)
                    self.mainPond.savePond(db,cursor)
                    self.window.close()
                    self.createUI()
                else:
                    error()
            self.createButton.clicked.connect(getInput)
            self.cancelButton.clicked.connect(self.window.close)

        def addFishWindow():
            self.window=QWidget()
            self.window.setWindowTitle("Pond Simulator 3000 - Add Fish")
            self.window.setFixedWidth(400)
            self.window.setFixedHeight(300)
            self.window.show()
            self.window.layout = QVBoxLayout()
            self.window.setLayout(self.window.layout)

            self.window.dropLayout=QGridLayout()
            self.window.widget=QWidget()
            self.window.widget.setLayout(self.window.dropLayout)
            self.window.dropLayout.addWidget(QLabel("Fish Type:"),0,0)
            self.window.dropDown=QComboBox()
            plantList=pondSimDatabase.fetchFishType(db,cursor)
            for count in range(0,len(plantList)):
                self.window.dropDown.addItem(plantList[count][0])
            self.window.dropLayout.addWidget(self.window.dropDown,0,1)
            self.window.layout.addWidget(self.window.widget)

            self.window.editWidget=QWidget()
            self.window.editLayout=QGridLayout()
            self.window.editWidget.setLayout(self.window.editLayout)
            self.nameEdit=QLineEdit()
            self.nameEdit.setMaxLength(10)
            self.window.editLayout.addWidget(QLabel("Fish Name:"),0,0)
            self.window.editLayout.addWidget(self.nameEdit,0,1)

            self.window.layout.addWidget(self.window.editWidget)

            self.window.widget=QWidget()

            def updateLabel():
                text=str(self.window.dropDown.currentText())
                if text!="Goldfish":
                    goldfishLabel.hide()
                else:
                    goldfishLabel.show()
                if text!="Rudd":
                    ruddLabel.hide()
                else:
                    ruddLabel.show()
                if text!="Tench":
                    tenchLabel.hide()
                else:
                    tenchLabel.show()
                if text!="Orfe":
                    orfeLabel.hide()
                else:
                    orfeLabel.show()
                if text!="Koi":
                    koiLabel.hide()
                else:
                    koiLabel.show()
                if text!="Shark":
                    sharkLabel.hide()
                else:
                    sharkLabel.show()

            def error():
                self.errorWindow=QWidget()
                self.errorWindow.setWindowTitle("Error")
                self.errorWindow.setFixedWidth(250)
                self.errorWindow.setFixedHeight(150)
                self.errorWindow.show()
                self.errorWindow.layout = QVBoxLayout()
                self.errorWindow.setLayout(self.errorWindow.layout)
                self.errorWindow.label=QLabel("You must open or create a pond before adding anything")
                self.errorWindow.label.setWordWrap(True)
                self.errorWindow.widget=QWidget()
                self.errorWindow.widgetLayout=QGridLayout()
                self.errorWindow.widget.setLayout(self.errorWindow.widgetLayout)
                self.errorWindow.okay=QPushButton("OK")
                self.errorWindow.okay.setFixedWidth(50)
                self.errorWindow.okay.clicked.connect(self.errorWindow.close)
                blank=QWidget()
                blank.setFixedWidth(100)
                blankTwo=QWidget()
                blankTwo.setFixedWidth(100)
                self.errorWindow.widgetLayout.addWidget(blank,0,2)
                self.errorWindow.widgetLayout.addWidget(blankTwo,0,0)
                self.errorWindow.widgetLayout.addWidget(self.errorWindow.okay,0,1)
                self.errorWindow.layout.addWidget(self.errorWindow.label)
                self.errorWindow.layout.addWidget(self.errorWindow.widget)

            def addPondFish():
```

```
            try:
                text=str(self.window.dropDown.currentText())
                name=self.nameEdit.text()
                self.mainPond.addFish(db,cursor,text,name)
                self.window.close()
                self.createUI()
            except:
                error()

        self.window.labelLayout=QVBoxLayout()
        self.window.widget.setLayout(self.window.labelLayout)

        goldfishLabel=QLabel("Rivalling Koi, are goldfish. They are possibly the most commonly kept garden fish, as they are small, inexpensive, colourful, and very hardy. They generally have a long lifespan. In a pond, they can often survive in icy weather conditions when ice forms on the surface, as long as there is enough oxygen remaining in the water and the pond does not freeze solid.")
        ruddLabel=QLabel("Rudd are a fantastic alternative to Orfe if you have a smaller pond. Rudds are acclimatised to the British weather and therefore are a hardy fish. They are a smaller growing surface fish which needs less oxygen. Rudd generally only achieve a size of 5-6 inches (12-15cm), however, they can get larger than this in large well filtered ponds.. Similar to Orfe, Rudd live much happier in small shoals. Rudd also like a varied diet, consisting of pellets, stick, and flaked dried preparations. They also like to feed from small larvae in the pond.")
        tenchLabel=QLabel("Tench are peaceful bottom feeding fish. . Tench are acclimatised to British weather. Although when living in lakes and large pools they grow to excess of 12 inches (30cm), rarely will they get as big as this in most garden ponds.Whilst it is a bit of a myth that tench will clean the bottom of your pond, they will disturb the silt and debris at the bottom, allowing the filter system to remove the waste more efficiently. Tench will thrive in well filtered large ponds, and are not ideally suited to small goldfish ponds. Tench are at their most happiest with sinking foods, and love scavenging around the bottom for worms and algae. The most common Tench are green, although golden and red can be purchased.")
        orfeLabel=QLabel("Orfe are fast growing, very active and a welcome addition to a garden pond. Although they are suitable for ponds, they do have slightly different requirements to goldfish. Orfe thrive well in faster moving water that is highly oxygenated. As they are a surface dwelling fish, therefore very oxygen dependant, ponds suitable for Orfe, should be well filtered, and oxygenated. Although they can survive in still water, they are much happier in moving water. Orfe can grow to lengths in excess of 24 inches (60cm), therefore they do require more space that goldfish. Orfe need to be kept in small shoals. They will feel much more secure, therefore the general health of the fish will be much improved. Orfe like a varied diet, consisting of pellets, stick, and flaked dried preparations. This fish will also be more than happy munching on small larvae in the pond.")
        koiLabel=QLabel("Koi fish are essentially an ornamental domesticated version of the common carp, and are sometimes known as ÃƒÂ¢Ã¢â€šÂ¬Ã‹Å“Japanese CarpÃƒÂ¢Ã¢â€šÂ¬Ã¢â€žÂ¢. They originate from Eastern Asia, Aral, Black and Caspian Seas. They are closely related to goldfish. A koi is considered a symbol of love and friendship. They are generally the most popular fish in the world to keep in ponds.Koi are freshwater, bottom dwelling fish, capable of living in a wide range of conditions. It is generally thought there are THIRTEEN different classifications of Koi. The Japanese classify koi according to various features, including colour, pattern, scale type and arrangement. However, within each classification there are different types of Koi. Koi come in many different colours and patterns, the more common colours are white, black, red, yellow, blue, and cream.")
        sharkLabel=QLabel("It's a shark. It will break your pond.")

        goldfishLabel.setWordWrap(True)
        ruddLabel.setWordWrap(True)
        tenchLabel.setWordWrap(True)
        orfeLabel.setWordWrap(True)
        koiLabel.setWordWrap(True)
        sharkLabel.setWordWrap(True)
```

```

            self.window.labelLayout.addWidget(goldfishLabel)
            self.window.labelLayout.addWidget(ruddLabel)
            self.window.labelLayout.addWidget(tenchLabel)
            self.window.labelLayout.addWidget(orfeLabel)
            self.window.labelLayout.addWidget(koiLabel)
            self.window.labelLayout.addWidget(sharkLabel)

            self.window.layout.addWidget(self.window.widget)
            updateLabel()
            self.window.dropDown.currentIndexChanged.connect(updateLabel)
            self.window.layout.addWidget(self.window.widget)

            self.window.buttonLayout=QGridLayout()
            self.window.buttonWidget=QWidget()
            self.window.buttonWidget.setLayout(self.window.buttonLayout)
            self.window.buttonLayout.addWidget(QWidget(),0,0)
            self.window.addButton=QPushButton("Add")
            self.window.cancelButton=QPushButton("Cancel")
            self.window.addButton.clicked.connect(addPondFish)
            self.window.cancelButton.clicked.connect(self.window.close)
            self.window.buttonLayout.addWidget(self.window.addButton,0,1)
            self.window.buttonLayout.addWidget(self.window.cancelButton,0,2)
            self.window.layout.addWidget(self.window.buttonWidget)

        def removeFishWindow():
            def tree():
                self.window.tree.setHeaderItem(self.window.header) #setting header
                pondID=self.mainPond.getPondID()
                fishTup=pondSimDatabase.fetchFish(db, cursor, pondID)
                for count in range(0,len(fishTup)):
                    self.window.treeList=QTreeWidgetItem()#list items
                    self.window.treeList.setText(0,str(fishTup[count][0]))
                    self.window.treeList.setText(1,str(fishTup[count][1]))
                    self.window.treeList.setText(2,str(fishTup[count][3]))
                    self.window.treeList.setText(3,str(fishTup[count][6]))
                    self.window.tree.addTopLevelItem(self.window.treeList)

            def onRemove():
                self.window.close() #closes window
                self.selected=self.window.tree.currentItem()
                ID=self.selected.data(0,0)
                pondSimDatabase.deleteFish(db, cursor, ID)
                self.createUI()

            self.window=QWidget()
            self.window.setWindowTitle("Pond Simulator 3000 - Open Pond")
            self.window.setFixedWidth(600)
            self.window.setFixedHeight(400)
            self.window.show()
            self.window.layout = QVBoxLayout()
            self.window.setLayout(self.window.layout)

            self.window.tree=QTreeWidget()
            self.window.tree.setColumnCount(4)
            self.window.tree.setColumnWidth(0,80)

            self.window.header=QTreeWidgetItem() #headers
            self.window.header.setText(0,'ID')
            self.window.header.setText(1,'Name')
            self.window.header.setText(2,'Status')
            self.window.header.setText(3,'Type')
            tree()

            self.window.button=QPushButton("Cancel") #making button
            self.window.button.setFixedWidth(100) #changing button size
            self.window.button.layout=QHBoxLayout() #creating layout for button
            self.window.buttonWidget=QWidget() #creating blank widget
            self.window.blank=QWidget()
            self.window.buttonWidget.setLayout(self.window.button.layout) #setting layout to blank widget
            self.window.button.layout.addWidget(self.window.blank) #adding button layout
            self.window.button.layout.addWidget(self.window.button) #adding button layout

            self.window.button.clicked.connect(self.window.close) #when button is clicked, window is hidden
            self.window.layout.addWidget(self.window.tree) #adding tree list to layout
            self.window.layout.addWidget(self.window.buttonWidget) #adding button to layout
            self.window.tree.itemDoubleClicked.connect(onRemove)#when item double clicked

        def addPlantWindow():
            self.window=QWidget()
            self.window.setWindowTitle("Pond Simulator 3000 - Add Plant")
            self.window.setFixedWidth(400)
            self.window.setFixedHeight(300)
            self.window.show()
            self.window.layout = QVBoxLayout()
            self.window.setLayout(self.window.layout)

            self.window.dropLayout=QGridLayout()
            self.window.widget=QWidget()
            self.window.widget.setLayout(self.window.dropLayout)
            self.window.dropLayout.addWidget(QLabel("Plant Type:"),0,0)
            self.window.dropDown=QComboBox()
            plantList=pondSimDatabase.fetchPlantType(db,cursor)
            for count in range(0,len(plantList)):
                self.window.dropDown.addItem(plantList[count][0])
            self.window.dropLayout.addWidget(self.window.dropDown,0,1)
            self.window.layout.addWidget(self.window.widget)

            def updateLabel():
                text=str(self.window.dropDown.currentText())
                if text!="Rooted Floater":
                    rootedLabel.hide()
                else:
                    rootedLabel.show()
                if text!="True Floater":
                    trueLabel.hide()
                else:
                    trueLabel.show()
                if text!="Emergent":
                    emergentLabel.hide()
                else:
                    emergentLabel.show()
                if text!="Submersed":
                    submersedLabel.hide()
                else:
                    submersedLabel.show()

            def error():
                self.errorWindow=QWidget()
                self.errorWindow.setWindowTitle("Error")
                self.errorWindow.setFixedWidth(250)
                self.errorWindow.setFixedHeight(150)
                self.errorWindow.show()
                self.errorWindow.layout = QVBoxLayout()
                self.errorWindow.setLayout(self.errorWindow.layout)
                self.errorWindow.label=QLabel("You must open or create a pond before adding anything")
                self.errorWindow.label.setWordWrap(True)
                self.errorWindow.widget=QWidget()
                self.errorWindow.widgetLayout=QGridLayout()
                self.errorWindow.widget.setLayout(self.errorWindow.widgetLayout)
                self.errorWindow.okay=QPushButton("OK")
                self.errorWindow.okay.setFixedWidth(50)
                self.errorWindow.okay.clicked.connect(self.errorWindow.close)
                blank=QWidget()
                blank.setFixedWidth(100)
                blankTwo=QWidget()
                blankTwo.setFixedWidth(100)
                self.errorWindow.widgetLayout.addWidget(blank,0,2)
                self.errorWindow.widgetLayout.addWidget(blankTwo,0,0)
                self.errorWindow.widgetLayout.addWidget(self.errorWindow.okay,0,1)
                self.errorWindow.layout.addWidget(self.errorWindow.label)
                self.errorWindow.layout.addWidget(self.errorWindow.widget)

            def addPondPlant():
                try:
                    text=str(self.window.dropDown.currentText())
                    self.mainPond.addPlant(db,cursor,text)
                    self.window.close()
                    self.createUI()
                except:
                    error()

            self.window.widget=QWidget()
            self.window.labelLayout=QVBoxLayout()
            self.window.widget.setLayout(self.window.labelLayout)

            rootedLabel=QLabel("1")
            trueLabel=QLabel("2")
            emergentLabel=QLabel("3")
            submersedLabel=QLabel("4")

            rootedLabel.setWordWrap(True)
            trueLabel.setWordWrap(True)
            emergentLabel.setWordWrap(True)
            submersedLabel.setWordWrap(True)

            self.window.labelLayout.addWidget(rootedLabel)
            self.window.labelLayout.addWidget(trueLabel)
            self.window.labelLayout.addWidget(emergentLabel)
            self.window.labelLayout.addWidget(submersedLabel)

            self.window.layout.addWidget(self.window.widget)
            updateLabel()
            self.window.dropDown.currentIndexChanged.connect(updateLabel)
            self.window.layout.addWidget(self.window.widget)

            self.window.buttonLayout=QGridLayout()
            self.window.buttonWidget=QWidget()
            self.window.buttonWidget.setLayout(self.window.buttonLayout)
            self.window.buttonLayout.addWidget(QWidget(),0,0)
            self.window.addButton=QPushButton("Add")
            self.window.cancelButton=QPushButton("Cancel")
            self.window.addButton.clicked.connect(addPondPlant)
            self.window.cancelButton.clicked.connect(self.window.close)
            self.window.buttonLayout.addWidget(self.window.addButton,0,1)
            self.window.buttonLayout.addWidget(self.window.cancelButton,0,2)
            self.window.layout.addWidget(self.window.buttonWidget)

#########
```

```

        def removePlantWindow():
            def tree():
                self.window.tree.setHeaderItem(self.window.header) #setting header
                pondID=self.mainPond.getPondID()
                plantTup=pondSimDatabase.fetchPlants(db, cursor, pondID)
                for count in range(0,len(plantTup)):
                    self.window.treeList=QTreeWidgetItem()#list items
                    self.window.treeList.setText(0,str(plantTup[count][0]))
                    self.window.treeList.setText(1,str(plantTup[count][2]))
                    self.window.treeList.setText(2,str(plantTup[count][1]))
                    self.window.tree.addTopLevelItem(self.window.treeList)

            def onRemove():
                self.window.close() #closes window
                self.selected=self.window.tree.currentItem()
                ID=self.selected.data(0,0)
                pondSimDatabase.deletePlant(db, cursor, ID)
                self.createUI()

            self.window=QWidget()
            self.window.setWindowTitle("Pond Simulator 3000 - Remove Plant")
            self.window.setFixedWidth(600)
            self.window.setFixedHeight(400)
            self.window.show()
            self.window.layout = QVBoxLayout()
            self.window.setLayout(self.window.layout)

            self.window.tree=QTreeWidget()
            self.window.tree.setColumnCount(3)
            self.window.tree.setColumnWidth(0,80)

            self.window.header=QTreeWidgetItem() #headers
            self.window.header.setText(0,'ID')
            self.window.header.setText(1,'Status')
            self.window.header.setText(2,'Type')
            tree()

            self.window.button=QPushButton("Cancel") #making button
            self.window.button.setFixedWidth(100) #changing button size
            self.window.button.layout=QHBoxLayout() #creating layout for button
            self.window.buttonWidget=QWidget() #creating blank widget
            self.window.blank=QWidget()
            self.window.buttonWidget.setLayout(self.window.button.layout) #setting layout to blank widget
            self.window.button.layout.addWidget(self.window.blank) #adding button layout
            self.window.button.layout.addWidget(self.window.button) #adding button layout

            self.window.button.clicked.connect(self.window.close) #when button is clicked, window is hidden
            self.window.layout.addWidget(self.window.tree) #adding tree list to layout
            self.window.layout.addWidget(self.window.buttonWidget) #adding button to layout
            self.window.tree.itemDoubleClicked.connect(onRemove)#when item double clicked


        def incrementDay():
            pond.incrementDay(self.mainPond,db,cursor)

        def editPondWindow():
            try:
                self.mainPond.updatePond(db,cursor)
                self.window=QWidget()
                self.window.setWindowTitle("Pond Simulator 3000 - Edit Pond")
                self.window.setFixedWidth(600)
                self.window.setFixedHeight(400)
                self.window.show()
                self.window.layout = QVBoxLayout()
                self.window.setLayout(self.window.layout)

                #box one
                self.window.mainWidget=QFrame()
                self.window.mainWidget.setLineWidth(1)
                self.window.mainWidget.setFrameShape(QFrame.StyledPanel)
                self.window.mainWidgetLayout=QVBoxLayout()
                self.window.mainWidget.setLayout(self.window.mainWidgetLayout)

                self.window.labelWidget=QWidget()
                self.window.labelWidgetLayout=QHBoxLayout()
                self.window.labelWidget.setLayout(self.window.labelWidgetLayout)
                self.window.labelWidgetLayout.addWidget(QLabel("The pond can be a maxium of 10 x 10 x 10 metres"))

                self.window.editWidget=QWidget()
                self.window.editWidgetLayout=QGridLayout()
                self.window.editWidget.setLayout(self.window.editWidgetLayout)
                self.window.editWidgetLayout.addWidget(QLabel("Depth:"),0,0)
                self.window.depthLine=QSpinBox()
                self.window.depthLine.setMaximum (10)
                self.window.depthLine.setMinimum(0.5)
                self.window.editWidgetLayout.addWidget(self.window.depthLine,0,1)
                self.window.editWidgetLayout.addWidget(QLabel("Width:"),0,2)
                self.window.widthLine=QSpinBox()
                self.window.widthLine.setMaximum (10)
                self.window.widthLine.setMinimum(0.5)
                self.window.editWidgetLayout.addWidget(self.window.widthLine,0,3)
                self.window.editWidgetLayout.addWidget(QLabel("Length:"),0,4)
                self.window.lengthLine=QSpinBox()
                self.window.lengthLine.setMaximum (10)
                self.window.lengthLine.setMinimum(0.5)
                self.window.editWidgetLayout.addWidget(self.window.lengthLine,0,5)

                self.window.mainWidgetLayout.addWidget(self.window.labelWidget)
                self.window.mainWidgetLayout.addWidget(self.window.editWidget)
                self.window.layout.addWidget(self.window.mainWidget)

                #box two
                self.window.mainWidget=QFrame()
                self.window.mainWidget.setLineWidth(1)
                self.window.mainWidget.setFrameShape(QFrame.StyledPanel)
                self.window.mainWidgetLayout=QVBoxLayout()
                self.window.mainWidget.setLayout(self.window.mainWidgetLayout)

                self.window.labelWidget=QWidget()
                self.window.labelWidgetLayout=QHBoxLayout()
                self.window.labelWidget.setLayout(self.window.labelWidgetLayout)
                self.window.labelWidgetLayout.addWidget(QLabel("The water temperature must be between 27 and +50C."))

                self.window.editWidget=QWidget()
                self.window.editWidgetLayout=QGridLayout()
                self.window.editWidget.setLayout(self.window.editWidgetLayout)
                self.window.editWidgetLayout.addWidget(QLabel("Temperature:"),0,0)
```

```
            self.window.tempLine=QSpinBox()
            self.window.tempLine.setMaximum (50)
            self.window.tempLine.setMinimum(-27)
            self.window.editWidgetLayout.addWidget(self.window.tempLine,0,1)
            self.window.blank=QWidget()
            self.window.blank.setFixedWidth(2000)
            self.window.editWidgetLayout.addWidget(self.window.blank,0,2)

            self.window.mainWidgetLayout.addWidget(self.window.labelWidget)
            self.window.mainWidgetLayout.addWidget(self.window.editWidget)
            self.window.layout.addWidget(self.window.mainWidget)

            #buttons
            self.createButton=QPushButton("Create")
            self.createButton.setFixedWidth(100)
            self.cancelButton=QPushButton("Cancel")
            self.cancelButton.setFixedWidth(100)
            self.blank=QWidget()
            self.blank.setFixedWidth(2000)
            self.buttonWidget=QWidget()
            self.buttonLayout=QGridLayout()
            self.buttonWidget.setLayout(self.buttonLayout)
            self.buttonLayout.addWidget(self.blank,0,0)
            self.buttonLayout.addWidget(self.createButton,0,1)
            self.buttonLayout.addWidget(self.cancelButton,0,2)
            self.window.layout.addWidget(self.buttonWidget)

            def error():
                self.errorWindow=QWidget()
                self.errorWindow.setWindowTitle("Error")
                self.errorWindow.setFixedWidth(250)
                self.errorWindow.setFixedHeight(150)
                self.errorWindow.show()
                self.errorWindow.layout = QVBoxLayout()
                self.errorWindow.setLayout(self.errorWindow.layout)
                self.errorWindow.label=QLabel("Length, width and depth values must all be greater than 0.")
                self.errorWindow.label.setWordWrap(True)
                self.errorWindow.widget=QWidget()
                self.errorWindow.widgetLayout=QGridLayout()
                self.errorWindow.widget.setLayout(self.errorWindow.widgetLayout)
                self.errorWindow.okay=QPushButton("OK")
                self.errorWindow.okay.setFixedWidth(50)
                self.errorWindow.okay.clicked.connect(self.errorWindow.close)
                blank=QWidget()
                blank.setFixedWidth(100)
                blankTwo=QWidget()
                blankTwo.setFixedWidth(100)
                self.errorWindow.widgetLayout.addWidget(blank,0,2)
                self.errorWindow.widgetLayout.addWidget(blankTwo,0,0)
                self.errorWindow.widgetLayout.addWidget(self.errorWindow.okay,0,1)
                self.errorWindow.layout.addWidget(self.errorWindow.label)
                self.errorWindow.layout.addWidget(self.errorWindow.widget)

            def getInput():
                #getting user inputs from screen
                depth=self.window.depthLine.text()
                length=self.window.lengthLine.text()
                width=self.window.widthLine.text()
                temp=self.window.tempLine.text()
                #setting inputs to pond
                if int(depth)!=0 and int(length)!=0 and int(width)!=0:
                    self.mainPond.setPondDepth(int(depth))
                    self.mainPond.setPondLength(int(length))
                    self.mainPond.setPondWidth(int(width))
                    self.mainPond.setWaterTemp(int(temp))
                    self.mainPond.setWaterLevel(int(depth)-0.5)

                    #saving pond
                    self.mainPond.updatePond(db,cursor)
                    self.window.close()
                    #updated GUI
                    self.createUI()
                else:
                    error() #error message displays if any of the dimentions are 0

           #button events
            self.createButton.clicked.connect(getInput)
            self.cancelButton.clicked.connect(self.window.close)

        except:
            self.errorWindow=QWidget()
            self.errorWindow.setWindowTitle("Error")
            self.errorWindow.setFixedWidth(250)
            self.errorWindow.setFixedHeight(150)
            self.errorWindow.show()
            self.errorWindow.layout = QVBoxLayout()
            self.errorWindow.setLayout(self.errorWindow.layout)
            self.errorWindow.label=QLabel("You must open a pond before you can edit it")
            self.errorWindow.label.setWordWrap(True)
            self.errorWindow.widget=QWidget()
            self.errorWindow.widgetLayout=QGridLayout()
            self.errorWindow.widget.setLayout(self.errorWindow.widgetLayout)
            self.errorWindow.okay=QPushButton("OK")
            self.errorWindow.okay.setFixedWidth(50)
            self.errorWindow.okay.clicked.connect(self.errorWindow.close)
            blank=QWidget()
            blank.setFixedWidth(100)
            blankTwo=QWidget()
            blankTwo.setFixedWidth(100)
            self.errorWindow.widgetLayout.addWidget(blank,0,2)
            self.errorWindow.widgetLayout.addWidget(blankTwo,0,0)
            self.errorWindow.widgetLayout.addWidget(self.errorWindow.okay,0,1)
            self.errorWindow.layout.addWidget(self.errorWindow.label)
            self.errorWindow.layout.addWidget(self.errorWindow.widget)


    def autoFeedWindow():
        pond.automaticGrow(db,cursor,self.mainPond)
        self.window=QWidget()
        self.window.setWindowTitle("Pond Simulator 3000 - Feed Fish")
        self.window.setFixedWidth(350)
        self.window.setFixedHeight(200)
        self.window.show()
        self.window.layout = QGridLayout()
        self.window.setLayout(self.window.layout)
        self.blank=QWidget()
        self.blank.setFixedWidth(130)
        self.window.layout.addWidget(self.blank,0,0)
        #self.window.layout.addWidget(self.blank,0,2)
        self.window.layout.addWidget(QLabel("Success!"),0,1)
        button=QPushButton("OK")
        button.clicked.connect(self.window.close)
        self.window.layout.addWidget(button,1,1)


        def manualFeedWindow():
            def manualGrow():
                amount=self.line.text()
                amount=int(amount)
                pond.manualGrow(amount,self.mainPond,db,cursor)
                self.window.close()

            self.window=QWidget()
            self.window.setWindowTitle("Pond Simulator 3000 - Feed Fish")
            self.window.setFixedWidth(600)
            self.window.setFixedHeight(400)
            self.window.show()
            self.window.layout = QVBoxLayout()
            self.window.setLayout(self.window.layout)

            totalHunger=0
            hunger=pondSimDatabase.fetchFishInfo(db,cursor,self.mainPond.getPondID())
            for count in range(0,len(hunger)):
                try:
                    totalHunger=hunger[count][2]+totalHunger
                except:
                    pass


            self.window.layout.addWidget(QLabel("Recommended amount to feed fish:"+str(totalHunger)))

            self.editWidget=QWidget()
            self.editWidget.layout=QGridLayout()
            self.editWidget.setLayout(self.editWidget.layout)
            self.editWidget.layout.addWidget(QLabel("Enter the amount of food you would like to add to the pond:"),0,0)
            self.line=QSpinBox()
            self.line.setMaximum (400)
            self.line.setMinimum(0)
            self.editWidget.layout.addWidget(self.line,0,1)
            self.window.layout.addWidget(self.editWidget)
            feedButton=QPushButton("Feed")
            cancelButton=QPushButton("Cancel")
            self.editWidget.layout.addWidget(feedButton,1,0)
            self.editWidget.layout.addWidget(cancelButton,1,1)
            feedButton.clicked.connect(manualGrow)
            cancelButton.clicked.connect(self.window.close)


#----------------------------------------------------------------------------------------------
        #setting title
        self.setWindowTitle('Pond Simulator 3000')

        #setting fixed screen size
        self.setFixedWidth(800)
        self.setFixedHeight(600)

        #creating menubar
        menu=QMenuBar()
        self.setMenuBar(menu)

        #adding menus
        fileMenu=menu.addMenu("File")
        manageMenu=menu.addMenu("Mange pond")

        #adding actions to file menu
        newAction=fileMenu.addAction("New Pond")
        openAction=fileMenu.addAction("Open Pond")
        saveAction=fileMenu.addAction("Save Pond")
        exitAction=fileMenu.addAction("Exit program")

        #adding actions to manage menu
        mpaf=manageMenu.addMenu("Manage pump and filter")
        eps=manageMenu.addAction("Edit pond settings")
        increment=manageMenu.addAction("Increment day")
        mpl=manageMenu.addMenu("Manage pond life")
        mpaf.addAction("Add pump and filter")
        mpaf.addAction("Remove pump and filter")
        mpaf.addAction("Eidt pump and filter")
        addFish=mpl.addAction("Add fish")
        addPlant=mpl.addAction("Add plant")
        removeFish=mpl.addAction("Remove fish")
        removePlant=mpl.addAction("Remove plant")
        ffm=mpl.addMenu("Feed fish")
        autoFeed=ffm.addAction("Automatically feed")
        manualFeed=ffm.addAction("Manually feed")

        #file menu action events
        exitAction.triggered.connect(application.closeAllWindows)
        openAction.triggered.connect(openWindow)
        saveAction.triggered.connect(savePond)
        newAction.triggered.connect(newPondWindow)

        #manage menu action events
        addFish.triggered.connect(addFishWindow)
        addPlant.triggered.connect(addPlantWindow)
        increment.triggered.connect(incrementDay)
        eps.triggered.connect(editPondWindow)
        removeFish.triggered.connect(removeFishWindow)
        removePlant.triggered.connect(removePlantWindow)
        autoFeed.triggered.connect(autoFeedWindow)
        manualFeed.triggered.connect(manualFeedWindow)

        #creating main layout
        self.layout = QGridLayout()
        self.mainWidget = QWidget()
        self.mainWidget.setLayout(self.layout)

        #creating graphics scenes for each widget
        self.meter=QGraphicsScene()
        self.graphics=QGraphicsScene()
        self.graphics.setSceneRect(0,0,550,400)
        self.scene4=QGraphicsScene()
        self.scene4.setSceneRect(0,0,550,150)
        #making each scene viewable
        self.meterView=QGraphicsView(self.meter)
        self.graphicsView=QGraphicsView(self.graphics)
        self.view4=QGraphicsView(self.scene4)

        #adding each viewable scene to its respective widget
        self.layout.addWidget(self.meterView,0,0)
        self.layout.addWidget(self.graphicsView,0,1)
        self.layout.addWidget(self.view4,1,1)

        #setting the main widget
        self.setCentralWidget(self.mainWidget)

#----------------GUI for graphical display---------------
        try:
            pump=pondSimDatabase.fetchPump(db,cursor,self.mainPond.getPump()) #fetching pump data
            pump=pond.Pump(pump[0][1], pump[0][0]) #creating pump object from pump data
            pumpStatus=pump.getPumpStatus()
        except:
            pumpStatus=""

        try:
            filter=pondSimDatabase.fetchFilter(db,cursor,self.mainPond.getFilter()) #fetching pump data
            filter=pond.Filter(filter[0][1], filter[0][0]) #creating pump object from pump data
            filterStatus=filter.getFilterStatus()
        except:
            filterStatus=""

        if filterStatus=="Broken":
            self.graphics.background=QPixmap("background2.png")
        else:
            self.graphics.background=QPixmap("background.png") #creates background
        self.graphics.backgroundItem=self.graphics.addPixmap(self.graphics.background)

        def addFish():
            fishResults=pondSimDatabase.fetchFish(db,cursor,self.mainPond.getPondID())
            for count in range(0,len(fishResults)):
                if fishResults[count][6]=="Goldfish":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
                    self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)
                    self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,399)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2

                elif fishResults[count][6]=="Tench":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
                    self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)
                    self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,399)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2

                elif fishResults[count][6]=="Orfe":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
                    self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)
                    self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,399)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2

                elif fishResults[count][6]=="Koi":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
                    self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)
```

```
self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,399))
#adds the fish graphic at a random location
                    self.graphics.fishItem.z=2

                elif fishResults[count][6]=="Rudd":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("fish.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("dead.png")
self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)
self.graphics.fishItem.setPos(random.randrange(1,549),random.randrange(1,399))
#adds the fish graphic at a random location
                    self.graphics.fishItem.z=2

                elif fishResults[count][6]=="Shark":
                    if fishResults[count][3]=="good":
                        self.graphics.fish=QPixmap("shark.png")
                    elif fishResults[count][3]=="Dead":
                        self.graphics.fish=QPixmap("sharkDead.png")
self.graphics.fishItem=self.graphics.addPixmap(self.graphics.fish)
                    self.graphics.fishItem.setPos(random.randrange(-
10,10),random.randrange(-20,20)) #adds the fish graphic at a random location
                    self.graphics.fishItem.z=2


        def addPlants():
plantResult=pondSimDatabase.fetchPlants(db,cursor,self.mainPond.getPondID())
            for count in range(0,len(plantResult)):
                if plantResult[count][1]=="Rooted Floater":
                    if plantResult[count][2]=="good":
                        self.graphics.plant=QPixmap("rootedFloater.png")
                    elif plantResult[count][2]=="Dead":
                        self.graphics.plant=QPixmap("rootedFloater.png")

                elif plantResult[count][1]=="True Floater":
                    if plantResult[count][2]=="good":
                        self.graphics.plant=QPixmap("trueFloater.png")
                    elif plantResult[count][2]=="Dead":
                        self.graphics.plant=QPixmap("trueFloater.png")

                elif plantResult[count][1]=="Emergent":
                    if plantResult[count][2]=="good":
                        self.graphics.plant=QPixmap("rootedFloater.png")
                    elif plantResult[count][2]=="Dead":
                        self.graphics.plant=QPixmap("rootedFloater.png")

                elif plantResult[count][1]=="Submersed":
                    if plantResult[count][2]=="good":
                        self.graphics.plant=QPixmap("rootedFloater.png")
                    elif plantResult[count][2]=="Dead":
                        self.graphics.plant=QPixmap("rootedFloater.png")


self.graphics.plantItem=self.graphics.addPixmap(self.graphics.plant)
                self.graphics.plantItem.setPos(random.randrange(1,549),0)
                self.graphics.plantItem.z=0

        try:
            addFish()
        except:
            pass

        try:
            addPlants()
        except:
            pass

#------------GUI for pump and filer status----------
        self.widgetF=QWidget() #creating widget
        try:
            pump=pondSimDatabase.fetchPump(db,cursor,self.mainPond.getPump()) #fetching pump data
            pump=pond.Pump(pump[0][1], pump[0][0]) #creating pump object from pump data
            pumpStatus=pump.getPumpStatus()
        except:
            pumpStatus=""

        try:
            filter=pondSimDatabase.fetchFilter(db,cursor,self.mainPond.getFilter()) #fetching pump data
            filter=pond.Filter(filter[0][1], filter[0][0]) #creating pump object from pump data
            filterStatus=filter.getFilterStatus()
        except:
            filterStatus=""

        self.pumpLabel=QLabel("Pump: "+pumpStatus)#making labels
        self.filterLabel=QLabel("Filter: "+filterStatus)
        self.layoutF=QGridLayout() #making layout
        self.widgetF.setLayout(self.layoutF) #giving widget layout
        self.layoutF.addWidget(self.pumpLabel,0,0) #adding labelsto layout
        self.layoutF.addWidget(self.filterLabel,1,0)
        self.layout.addWidget(self.widgetF,1,0) #adding widget to main layout

#-----------------GUI for meters--------------------
        #water meter outline
        self.meter.waterMeter=QRectF(10,-50,50,200) #creating rect
        self.meter.waterMeter.pen=QPen() #creating pen
        self.meter.waterMeter.color=QColor() #creating colour for pen
        self.meter.waterMeter.color.setNamedColor('#DCDCDC')
        self.meter.waterMeter.pen.setColor(self.meter.waterMeter.color) #setting color of pen
        self.meter.waterMeter.pen.setWidth(2) #setting line width
        self.meter.waterMeter.brush=QBrush() #creating blank fill
        self.meter.addRect(self.meter.waterMeter,self.meter.waterMeter.pen,self.meter.waterMeter.brush) #drawing rect

        #temperature meter outline
        self.meter.tempMeter=QRectF(120,-50,50,200)
        self.meter.tempMeter.pen=QPen()
        self.meter.tempMeter.color=QColor() #creating colour for pen
        self.meter.tempMeter.color.setNamedColor('#E04006')
        self.meter.tempMeter.pen.setColor(self.meter.tempMeter.color) #setting color of pen
        self.meter.tempMeter.pen.setWidth(2) #setting line width
        self.meter.tempMeter.brush=QBrush()
        self.meter.addRect(self.meter.tempMeter,self.meter.tempMeter.pen,self.meter.tempMeter.brush)


        #water meter fill
        try:
            number=float(self.mainPond.getWaterLevel())
            number=number*20
        except:
            number=0
        self.meter.waterMeterFill=QRectF(11,-50,47,number)
        self.meter.waterMeterFill.moveBottom(148)
        self.meter.waterMeterFill.pen=QPen() #creating pen
        self.meter.waterMeterFill.color=QColor() #creating colour for pen
        self.meter.waterMeterFill.color.setNamedColor('#FFFFFF')
        self.meter.color=QColor()
        self.meter.color.setNamedColor('lightseagreen')
        self.meter.waterMeterFill.pen.setColor(self.meter.waterMeterFill.color) #setting color of pen
        self.meter.waterMeterFill.brush=QBrush() #creating blank fill
        self.meter.waterMeterFill.brush.setColor(self.meter.color)
        self.meter.waterMeterFill.brush.setStyle(Qt.SolidPattern)
        self.meter.addRect(self.meter.waterMeterFill,self.meter.waterMeterFill.pen,self.meter.waterMeterFill.brush) #drawing rect

        #water temperature fill
        try:
            number=float(self.mainPond.getWaterTemp())
            if number>=0:
                number=number*2
            else:
                number=0
        except:
            number=0
        self.meter.tempMeterFill=QRectF(122,-50,45,number)
        self.meter.tempMeterFill.moveBottom(48)
        self.meter.tempMeterFill.pen=QPen()#creating pen
        color=QColor() #creating colour for pen
        color.setNamedColor('#FFFFFF')
        self.meter.tempMeterFill.color=QColor()
        self.meter.tempMeterFill.color.setNamedColor('red')
        self.meter.tempMeterFill.pen.setColor(self.meter.tempMeterFill.color) #setting color of pen
        self.meter.tempMeterFill.brush=QBrush() #creating blank fill
        self.meter.tempMeterFill.brush.setColor(self.meter.tempMeterFill.color)
        self.meter.tempMeterFill.brush.setStyle(Qt.SolidPattern)
        self.meter.addRect(self.meter.tempMeterFill,self.meter.tempMeterFill.pen,self.meter.tempMeterFill.brush) #drawing rect

        #water temperature minus fill
        try:
            number=float(self.mainPond.getWaterTemp())
            if number<0:
                number=-number*2
            else:
                number=100
        except:
            number=0
        self.meter.tempMeterFill=QRectF(122,-50,45,number)
        self.meter.tempMeterFill.moveBottom(147)
        self.meter.tempMeterFill.pen=QPen() #creating pen
        color=QColor() #creating colour for pen
        color.setNamedColor('#FFFFFF')
        self.meter.tempMeterFill.color=QColor()
        self.meter.tempMeterFill.color.setNamedColor('blue')
        self.meter.tempMeterFill.pen.setColor(self.meter.tempMeterFill.color) #setting color of pen
        self.meter.tempMeterFill.brush=QBrush() #creating blank fill
        self.meter.tempMeterFill.brush.setColor(self.meter.tempMeterFill.color)
        self.meter.tempMeterFill.brush.setStyle(Qt.SolidPattern)
        self.meter.addRect(self.meter.tempMeterFill,self.meter.tempMeterFill.pen,self.meter.tempMeterFill.brush) #drawing rect

        self.widgetM=QWidget() #creating widget
        try:
            level=str(self.mainPond.getWaterLevel())
        except:
            level=""
        try:
            temp=str(self.mainPond.getWaterTemp())
        except:
            temp=""
        self.waterLabel=QLabel("Water level: "+level)#making labels
        self.tempLabel=QLabel("Temperature: "+temp)
        self.zeroLabel=QLabel("  0")
        self.zeroLabel.setFixedHeight(8)
        self.fiftyLabel=QLabel("  50")
        self.fiftyLabel.setFixedHeight(185)
        self.sevenLabel=QLabel("  -27")
        self.sevenLabel.setFixedHeight(160)
        self.layoutM=QGridLayout() #making layout
        self.widgetM.setLayout(self.layoutM) #giving widget layout
        self.layoutM.addWidget(self.waterLabel,3,0) #adding labels to layout
        self.layoutM.addWidget(self.tempLabel,3,1)
        self.layoutM.addWidget(self.fiftyLabel,0,1)
        self.layoutM.addWidget(self.zeroLabel,1,1)
        self.layoutM.addWidget(self.sevenLabel,2,1)
        self.layout.addWidget(self.widgetM,0,0) #adding widget to main layout

#---------------------GUI for about the pond-------------------
        try:
            fishList=pondSimDatabase.fetchFish(db,cursor,self.mainPond.getPondID())
            plantList=pondSimDatabase.fetchPlants(db,cursor,self.mainPond.getPondID())

            goldfishTotal=0
            tenchTotal=0
            orfeTotal=0
            koiTotal=0
            ruddTotal=0
            sharkTotal=0

            for count in range(0,len(fishList)):
                if fishList[count][6]=="Goldfish":
                    goldfishTotal+=1
                elif fishList[count][6]=="Tench":
                    tenchTotal+=1
                elif fishList[count][6]=="Orfe":
                    orfeTotal+=1
                elif fishList[count][6]=="Koi":
                    koiTotal+=1
                elif fishList[count][6]=="Rudd":
                    ruddTotal+=1
                elif fishList[count][6]=="Shark":
                    sharkTotal+=1

            self.scene4.goldfish=QGraphicsTextItem("There are "+str(goldfishTotal)+" Goldfish in the pond")
            self.scene4.tench=QGraphicsTextItem("There are "+str(tenchTotal)+" tench in the pond")
            self.scene4.orfe=QGraphicsTextItem("There are "+str(orfeTotal)+" orfe in the pond")
            self.scene4.koi=QGraphicsTextItem("There are "+str(koiTotal)+" koi in the pond")
            self.scene4.rudd=QGraphicsTextItem("There are "+str(ruddTotal)+" rudd in the pond")
            self.scene4.shark=QGraphicsTextItem("There are "+str(sharkTotal)+" sharks in the pond")

            self.scene4.goldfish.setPos(20,5)
            self.scene4.tench.setPos(20,22)
            self.scene4.orfe.setPos(20,39)
            self.scene4.koi.setPos(20,56)
            self.scene4.rudd.setPos(20,73)
            self.scene4.shark.setPos(20,90)

            self.scene4.addItem(self.scene4.goldfish)
            self.scene4.addItem(self.scene4.tench)
            self.scene4.addItem(self.scene4.orfe)
            self.scene4.addItem(self.scene4.koi)
            self.scene4.addItem(self.scene4.rudd)
            self.scene4.addItem(self.scene4.shark)

            emergentTotal=0
            rootedTotal=0
            trueTotal=0
            submersedTotal=0

            for count in range(0,len(plantList)):
                if plantList[count][1]=="Emergent":
                    emergentTotal+=1
                elif plantList[count][1]=="Rooted Floater":
                    rootedTotal+=1
                elif plantList[count][1]=="True Floater":
                    trueTotal+=1
                elif plantList[count][1]=="Submersed":
                    submersedTotal+=1

            self.scene4.emergent=QGraphicsTextItem("There are "+str(emergentTotal)+" emergent plants in the pond")
            self.scene4.rooted=QGraphicsTextItem("There are "+str(rootedTotal)+" rooted plants in the pond")
            self.scene4.true=QGraphicsTextItem("There are "+str(trueTotal)+" floating plants in the pond")
            self.scene4.submersed=QGraphicsTextItem("There are "+str(submesedTotal)+" submersed plants in the pond")

            self.scene4.emergent.setPos(220,5)
            self.scene4.rooted.setPos(220,22)
            self.scene4.true.setPos(220,39)
            self.scene4.submersed.setPos(220,56)

            self.scene4.addItem(self.scene4.emergent)
            self.scene4.addItem(self.scene4.rooted)
            self.scene4.addItem(self.scene4.true)
            self.scene4.addItem(self.scene4.submersed)

        except:
            pass


if __name__=="__main__":
    db=sqlite3.connect('pond.db') #connecting to database
    cursor=db.cursor()
    databaseCreation.createDatabases()
    fish.saveFishTypes(db,cursor)
    plant.savePlantTypes(db,cursor)

    application=QApplication(sys.argv) #create application
    mainWindow=MainWindow() #creat new window instance
    mainWindow.show() #make instance visable
    mainWindow.raise_() #raise instance to top of window stack
    application.exec_() #monitor for events
```

10.3 Fish Module

```
import sqlite3

class Fish:
    """A simulation of a generic fish"""
    def __init__(self, db, cursor):
        self._fishName=None
        self._fishType="fish"
        self._foodNeed=0
        self._growthRate=0
        self._fishSize=0
        self._hunger=0
        self._status="good"
        self._daysAlive=0
        self._growth=0
        self._maxDaysAlive=100
        self._fishID=None

    def getFishID(self):
        return self._fishID

    def setFishName(self,x):
        self._fishName=x

    def getFishName(self):
        return self._fishName

    def setFishType(self,x):
        self._fishType=x

    def getFishType(self):
        return self._fishType

    def setFoodNeed(self,x):
        self._foodNeed=x

    def getFoodNeed(self):
        return self._foodNeed

    def setGrowthRate(self,x):
        self._growthRate=x

    def getGrowthRate(self):
        return self._growthRate

    def setFishSize(self,x):
        self._setFishSize=x

    def getFishSize(self):
        return self._fishSize

    def setHunger(self,x):
        self._hunger=x

    def getHunger(self):
        return self._hunger

    def setStatus(self,growth,daysAlive):
        pass

    def getStatus(self):
        return self._status

    def setDaysAlive(self,x):
        self._daysAlive=x

    def getDaysAlive(self):
        return self._daysAlive

    def setGrowth(self,x):
        self._growth=x

    def getGrowth(self):
        return self._growth

    def grow(self,food):
        pass

    def saveFish(self, pondID, db, cursor):
        sql = """insert into fish(fishName, hunger, status, daysAlive, growth, fishType, pondID) values
                ('{0}','{1}','{2}','{3}','{4}','{5}','{6}')""".format(self._fishName, self._hunger, self._status, self._daysAlive, self._growth, self._fishType, pondID)
        cursor.execute(sql)
        cursor.execute("""select last_insert_rowid()""")
        self._fishID = cursor.fetchall()[0][0]
        db.commit()

class Goldfish(Fish):
    """A goldfish fish type"""
    def __init__(self, db, cursor):
        Fish.__init__(self, db, cursor)
        self._fishType="Goldfish"
        self._foodNeed=5
        self._growthRate=1
        self._fishSize=0.5
    def save(self, db, cursor):
        sql = """insert into fishType(fishType,foodNeed, growthRate, fishSize) values
            ('{0}','{1}','{2}','{3}')""".format(self._fishType, self._foodNeed, self._growthRate, self._fishSize)
        cursor.execute(sql)
        db.commit()

class Tench(Fish):
    """A tench fish type"""
    def __init__(self, db, cursor):
        Fish.__init__(self, db, cursor)
        self._fishType="Tench"
        self._foodNeed=4
        self._growthRate=0.5
        self._fishSize=1
    def save(self, db, cursor):
        sql = """insert into fishType(fishType,foodNeed, growthRate, fishSize) values
            ('{0}','{1}','{2}','{3}')""".format(self._fishType, self._foodNeed, self._growthRate, self._fishSize)
        cursor.execute(sql)
        db.commit()

class Orfe(Fish):
    """A Orfe fish type"""
    def __init__(self, db, cursor):
        Fish.__init__(self, db, cursor)
        self._fishType="Orfe"
        self._foodNeed=3
        self._growthRate=3
        self._fishSize=1
    def save(self, db, cursor):
        sql = """insert into fishType(fishType,foodNeed, growthRate, fishSize) values
```

```
            ('{0}','{1}','{2}','{3}')""".format(self._fishType, self._foodNeed, self._growthRate, self._fishSize)
        cursor.execute(sql)
        db.commit()

class Koi(Fish):
    """A koi fish type"""
    def __init__(self, db, cursor):
        Fish.__init__(self, db, cursor)
        self._fishType="Koi"
        self._foodNeed=8
        self._growthRate=2
        self._fishSize=0.5
    def save(self, db, cursor):
        sql = """insert into fishType(fishType,foodNeed, growthRate, fishSize) values
            ('{0}','{1}','{2}','{3}')""".format(self._fishType, self._foodNeed, self._growthRate, self._fishSize)
        cursor.execute(sql)
        db.commit()

class Rudd(Fish):
    """A rudd fish type"""
    def __init__(self, db, cursor):
        Fish.__init__(self, db, cursor)
        self._fishType="Rudd"
        self._foodNeed=2
        self._growthRate=0.5
        self._fishSize=0.25
    def save(self, db, cursor):
        sql = """insert into fishType(fishType,foodNeed, growthRate, fishSize) values
            ('{0}','{1}','{2}','{3}')""".format(self._fishType, self._foodNeed, self._growthRate, self._fishSize)
        cursor.execute(sql)
        db.commit()

class Shark(Fish):
    """A shark fish type"""
    def __init__(self, db, cursor):
        Fish.__init__(self, db, cursor)
        self._fishType="Shark"
        self._foodNeed=1000
        self._growthRate=1
        self._fishSize=100
    def save(self, db, cursor):
        sql = """insert into fishType(fishType,foodNeed, growthRate, fishSize) values
            ('{0}','{1}','{2}','{3}')""".format(self._fishType, self._foodNeed, self._growthRate, self._fishSize)
        cursor.execute(sql)
        db.commit()


def saveFishTypes(db,cursor):
    newGold=Goldfish(db,cursor)
    newTench=Tench(db,cursor)
    newOrfe=Orfe(db,cursor)
    newKoi=Koi(db,cursor)
    newRudd=Rudd(db,cursor)
    newShark=Shark(db,cursor)
    try:
        newGold.save(db, cursor)
        newTench.save(db, cursor)
        newOrfe.save(db, cursor)
        newKoi.save(db, cursor)
        newRudd.save(db, cursor)
        newShark.save(db, cursor)
    except:
        pass


if __name__ == "__main__":
    db=sqlite3.connect('pond.db')
    cursor=db.cursor()
    saveFishTypes(db,cursor)
    cursor.close()
```

10.4 Plant Module

```
import sqlite3

class Plant:
    """A simulation of a generic plant, requires a plantType object"""
    def __init__(self, db, cursor):
        self._status="good"
        self._lightNeed=0
        self._daysAlive=0
        self._growth=0
        self._plantID=None
        self._plantType="Plant"

    def getPlantID(self):
        return self._plantID

    def setPlantType(self,x):
        self._plantType=x

    def getPlantType(self):
        return self._plantType

    def setStatus(self,x):
        self._status=x

    def getStatus(self):
        return self._status

    def setLightNeed(self,x):
        self._lightNeed=x

    def getLightNeed(self):
        return self._lighNeed

    def setDaysAlive(self,x):
        self._daysAlive=x

    def getDaysAlive(self):
        return self._daysAlive

    def setGrowth(self,x):
        self._growth=x

    def getGrowth(self):
        return self._growth

    def savePlant(self, pondID, db,cursor):
        sql = """insert into plant(status, lightNeed, daysAlive, growth,
plantType, PondID) values
                ( '{0}','{1}','{2}','{3}','{4}','{5}')""".format(self._status,
self._lightNeed, self._daysAlive, self._growth, self._plantType, pondID)
        cursor.execute(sql)
        cursor.execute("""select last_insert_rowid()""")
        self._plantID = cursor.fetchall()[0][0]
        db.commit()

class RootedFloater(Plant):
    """A rooted floater plant type"""
    def __init__(self,db,cursor):
        Plant.__init__(self,db,cursor)
        self._plantType="Rooted Floater"
        self._lightRequirement=7
        self._growthRate=2
    def save(self, db, cursor):
        sql = """insert into plantType(plantType, lightRequirement, growthRate) values
            ('{0}','{1}','{2}')""".format(self._plantType, self._lightRequirement, self._growthRate)
        cursor.execute(sql)
        db.commit()

class TrueFloater(Plant):
    """A true floater plant type"""
    def __init__(self,db,cursor):
        Plant.__init__(self,db,cursor)
        self._plantType="True Floater"
        self._lightRequirement=2
        self._growthRate=2
    def save(self, db, cursor):
        sql = """insert into plantType(plantType, lightRequirement, growthRate) values
            ('{0}','{1}','{2}')""".format(self._plantType, self._lightRequirement, self._growthRate)
        cursor.execute(sql)
        db.commit()

class Emergent(Plant):
    """An emergent plant type"""
    def __init__(self,db,cursor):
        Plant.__init__(self,db,cursor)
        self._plantType="Emergent"
        self._lightRequirement=2
        self._growthRate=1
    def save(self, db, cursor):
        sql = """insert into plantType(plantType, lightRequirement, growthRate) values
            ('{0}','{1}','{2}')""".format(self._plantType, self._lightRequirement, self._growthRate)
        cursor.execute(sql)
        db.commit()

class Submersed(Plant):
    """A Submersed plant type"""
    def __init__(self,db,cursor):
        Plant.__init__(self,db,cursor)
        self._plantType="Submersed"
        self._lightRequirement=2
        self._growthRate=1
    def save(self, db, cursor):
        sql = """insert into plantType(plantType, lightRequirement, growthRate) values
            ('{0}','{1}','{2}')""".format(self._plantType, self._lightRequirement, self._growthRate)
        cursor.execute(sql)
        db.commit()

def savePlantTypes(db, cursor):
    newRooted=RootedFloater(db,cursor)
    newFloat=TrueFloater(db, cursor)
    newEmergent=Emergent(db,cursor)
    newSubmersed=Submersed(db,cursor)
    try:
        newRooted.save(db, cursor)
        newFloat.save(db, cursor)
        newEmergent.save(db, cursor)
        newSubmersed.save(db, cursor)
    except:
        pass


if __name__ == "__main__":
    db=sqlite3.connect('pond.db')
    cursor=db.cursor()
    savePlantTypes(db, cursor)
    cursor.close()
```

10.5 Pond Module

```
import sqlite3
import pondSimDatabase
import plant
import fish
import databaseCreation
import random

#classes
class Pond:
    """A simulation of a pond"""
    def __init__(self, width, length, depth, temp, waterLevel, pump, filter, pondID, days):
        self._pondWidth=width
        self._pondLength=length
        self._pondDepth=depth
        self._waterTemp=temp
        self._waterLevel=waterLevel
        self._pump=pump
        self._filter=filter
        self._pondID=pondID
        self._days=days

    def setPondID(self,x):
        self._pondID=x

    def getPondID(self):
        return self._pondID

    def setPondWidth(self, x): #where x is the desired pond width
        self._pondWidth=x

    def getPondWidth(self):
        return self._pondWidth

    def setPondLength(self, x):
        self._pondLength=x

    def getPondLength(self):
        return self._pondLength

    def setPondDepth(self,x):
        self._pondDepth=x

    def getPondDepth(self):
        return self._pondDepth

    def setWaterLevel(self,x):
        self._waterLevel=x

    def getWaterLevel(self):
        return self._waterLevel

    def setWaterTemp(self, x):
        self._waterTemp=x

    def getWaterTemp(self):
        return self._waterTemp

    def setPump(self, x): #where x is the pump object
        self._pump=x

    def getPump(self):
        return self._pump

    def setFilter(self, x):
        self._filter=x

    def getFilter(self):
        return self._filter

    def setDays(self, x):
        self._days=x

    def getDays(self):
        return self._days

    def addFish(self, db, cursor,choice,name):
        def chooseFish(db, cursor,choice):
            if choice=="Goldfish":
                newFish=fish.Goldfish(db, cursor)
            elif choice=="Tench":
                newFish=fish.Tench(db,cursor)
            elif choice=="Orfe":
                newFish=fish.Orfe(db,cursor)
            elif choice=="Koi":
                newFish=fish.Koi(db,cursor)
            elif choice=="Rudd":
                newFish=fish.Rudd(db,cursor)
            elif choice=="Shark":
                newFish=fish.Shark(db,cursor)
            return newFish

        newFish=chooseFish(db, cursor,choice)
        newFish.setFishName(name)
        newFish.saveFish(self._pondID, db, cursor)

    def removeFish(self,db, cursor):
        pondID=pond.getPondID()
        fishTup=pondSimDatabase.fetchFish(db, cursor, pondID)
        correct=False

        #displaying list of fish in pond
print('|{0:^10}|{1:^10}|{2:^15}|{3:^15}|{4:^15}|{5:^15}|{6:^10}|'.format('FishID','fishName','Hunger','Status','daysAlive','growth','FishType'))
        for each in range(0,len(fishTup)):
            print('{0:^12} {1:^10} {2:^10} {3:^20} {4:^10} {5:^20} {6:^5}'.format(fishTup[each][0],fishTup[each][1],fishTup[each][2],fishTup[each][3],fishTup[each][4],fishTup[each][5],fishTup[each][6]))
        print('')

        try:
            while correct==False:
                IDchoice=int(input('Enter the ID of the fish you wish to remove: '))
                if IDchoice>len(fishTup)or IDchoice<0:
                    print("That ID was not found on the list")
                else:
                    correct=True
                    pondSimDatabase.deleteFish(db, cursor, IDchoice)
        except:
            print("That was not a valid ID - please enter a number")


    def removePlant(self,db,cursor):
        pondID=pond.getPondID()
        plantsTup=pondSimDatabase.fetchPlants(db, cursor, pondID)
        correct=False


print('|{0:^10}|{1:^10}|{2:^15}|{3:^15}|{4:^15}|{5:^15}|'.format('PlantID','PlantTy
pe','Status','LightNeed','daysAlive','growth'))
        for each in range(0,len(plantsTup)):
            print('{0:^12} {1:^10} {2:^10} {3:^20} {4:^10}
{5:^20}'.format(plantsTup[each][0],plantsTup[each][1],plantsTup[each][2],plantsTup[
each][3],plantsTup[each][4],plantsTup[each][5]))
        print('')

        try:
            while correct==False:
                IDchoice=int(input('Enter the ID of the plant you wish to
remove: '))
                if IDchoice>len(plantTup)or IDchoice<0:
                    print("That ID was not found on the list")
                else:
                    correct=True
                    pondSimDatabase.deletePlant(db, cursor, IDchoice)
        except:
            print("That was not a valid ID - please enter a number")

    def addPlant(self, db, cursor,choice):
        def choosePlant(db,cursor,choice):
            if choice=="Rooted Floater":
                newPlant=plant.RootedFloater(db,cursor)
            elif choice=="True Floater":
                newPlant=plant.TrueFloater(db,cursor)
            elif choice=="Emergent":
                newPlant=plant.Emergent(db,cursor)
            elif choice=="Submersed":
                newPlant=plant.Submersed(db,cursor)
            return newPlant
        newPlant=choosePlant(db,cursor,choice)
        newPlant.savePlant(self._pondID, db, cursor)

    def editPond(self):
        waterTemp=input("Water Temperature:")
        self.setWaterTemp(waterTemp)

        pondWidth=input("Pond Width:")
        self.setPondWidth(pondWidth)

        pondLength=input("Pond Length:")
        self.setPondLength(pondLength)

        pondDepth=input("Pond Depth:")
        self.setPondDepth(pondDepth)

        self.setWaterLevel(int(pondDepth)-0.5)

    def savePond(self,db,cursor):
        #saving pond
        try:
            self._pumpID=self._pump.getPumpID()
        except:
            self._pumpID=None

        try:
            self._filterID=self._filter.getFilterID()
        except:
            self._filterID=None

        sql = """insert into pond(pondDepth, pondLength, pondWidth, waterTemp, waterLevel, pumpID, filterID, days) values
                 ( '{0}','{1}','{2}','{3}','{4}','{5}','{6}','{7}')""".format(self._pondDepth, self._pondLength, self._pondWidth, self._waterTemp, self._waterLevel, self._pump, self._filter, self._days)
        cursor.execute(sql)
        cursor.execute("""select last_insert_rowid()""")
        self._pondID = cursor.fetchall()[0][0]
        db.commit()

    def updatePond(self,db,cursor):
        try:
            self._pumpID=self.getPump()
        except:
            self._pumpID=None

        try:
            self._filterID=self.getFilter()
        except:
            self._filterID=None
        try:
            tup=(self._pondDepth, self._pondLength, self._pondWidth, self._waterTemp, self._waterLevel, self._days, self._pumpID, self._filterID, self._pondID)
            sql = """update pond
                     set pondDepth=?, pondLength=?, pondWidth=?, waterTemp=?, waterLevel=?, days=?, pumpID=?, filterID=?
                     where pondID=?"""
            cursor.execute(sql,tup)
            db.commit()
        except:
            pass

class Filter:
    """A simulation of a pond filter"""
    def __init__(self, status, filterID):
        self._filterStatus=status
        self._filterID=filterID

    def getFilterID(self):
        return self._filterID

    def setFilterStatus(self,x):
        self._filterStatus=x

    def getFilterStatus(self):
        return self._filterStatus

    def saveFilter(self,db,cursor):
        sql = """insert into filter(filterStatus) values
                 ( '{0}')""".format(self._filterStatus)
        cursor.execute(sql)
        cursor.execute("""select last_insert_rowid()""")
        self._filterID = cursor.fetchall()[0][0]
        db.commit()

    def updateFilter(self,db,cursor):
        tup=(self._filterStatus,self._filterID)
        sql="""update filter
           set filterStatus=?
           where filterID=?"""
        cursor.execute(sql,tup)
        db.commit()

class Pump:
    """A simulation of a pump"""
    def __init__(self, status, pumpID):
        self._pumpStatus=status
        self._pumpID=pumpID

    def getPumpID(self):
        return self._pumpID

    def setPumpStatus(self,x):
        self._pumpStatus=x

    def getPumpStatus(self):
        return self._pumpStatus

    def savePump(self,db,cursor):
        sql = """insert into pump(pumpStatus) values
                ( '{0}')""".format(self._pumpStatus)
        cursor.execute(sql)
        cursor.execute("""select last_insert_rowid()""")
        self._pumpID = cursor.fetchall()[0][0]
        db.commit()

    def updatePump(self,db,cursor):
        tup=(self._pumpStatus,self._pumpID)
        sql="""update pump
           set pumpStatus=?
           where pumpID=?"""
        cursor.execute(sql,tup)
        db.commit()

#creating a pond
def createPond():
    width=0
    while width>10 or width<1:
        try:
            width=int(input('Enter the width of the pond: '))
            if width>10 or width<1:
                print("The width of the pond must be between 1 and 10 (1 = 1m)")
        except:
            print("The width of the pond must be a number between 1 and 10 (1 = 1m)")
    print('')

    length=0
    while length>10 or length<1:
        try:
            length=int(input('Enter the length of the pond: '))
            if length>10 or length<1:
                print("The length of the pond must be between 1 and 10 (1 = 1m)")
        except:
            print("The length of the pond must be a number between 1 and 10 (1 = 1m)")
    print('')

    depth=0
    while depth>10 or depth<1:
        try:
            depth=int(input('Enter the depth of the pond: '))
            if depth>10 or depth<1:
                print("The depth of the pond must be between 1 and 10 (1 = 1m)")
        except:
            print("The depth of the pond must be a number between 1 and 10 (1 = 1m)")
    print('')

    temp=100
    while temp>50 or temp<-27:
        try:
            temp=int(input('Enter the starting temperature of the pond: '))
            if temp>50 or temp<-27:
                print("The water temperature must be between -27C")
        except:
            print("The water temperature must be a number between -27C")
    print('')

    pandf=4
    while pandf>1 or pandf<0:
        try:
            pandf=int(input('Press 1 to add a pump and filter to the pond or 0 to not: '))
            if pandf>1 or pandf<0:
                print("The value entered was not 1 or 0. Try again")
        except:
            print("The value entered was not 1 or 0. Try again")
    print('')

    waterLevel=float(depth)-0.5
    if int(pandf)==1:
        pump=Pump("good",None)
        pump.savePump()
        filter=Filter("good",None)
        filter.saveFilter()
    else:
        pump=None
        filter=None
    newPond=Pond(width, length, depth, temp, waterLevel, pump, filter,None,0)
    newPond.savePond()
    return newPond

#main menu - first thing displayed
def mainMenu():
    def displayMenu():
        print('')
        print("1. Create new pond")
        print("2. Open pond")
        print("3. Delete Pond")
        print("4. Exit program")
        print('')

    def getMenuChoice():
        choice=0
        while choice==0:
            try:
                choice=int(input("What would you like to do? "))
                if choice>4 or choice<1:
                    print("You must enter an option between 1 and 4")
            except:
                print("You must enter a number between 1 and 4")
        return choice

    displayMenu()
    choice=getMenuChoice()
    return choice

#openning an existing pond
```

```
def openPond():
    #fetching and displaying all ponds in database
    correct=False
    results=pondSimDatabase.getPonds(db,cursor)
    #getting ID from user
    pondIDs=pondSimDatabase.fetchPondID(db,cursor)

    try:
        while correct==False:
            IDchoice=int(input('Enter the ID of the pond you wish to open: '))
            for count in range(0,len(pondIDs)):
                if IDchoice==pondIDs[count][0]:
                    correct=True
            if correct==False:
                print("That ID was not found on the list")
    except:
        print("That was not a valid ID - please enter a number")

    results=pondSimDatabase.fetchPond(db,cursor,str(IDchoice))#fetching pond data
    try:
        pump=pondSimDatabase.fetchPump(db,cursor,str(results[0][6])) #fetching pump data
        pump=Pump(pump[0][1], pump[0][0]) #creating pump object from pump data
        filter=pondSimDatabase.fetchFilter(db,cursor,str(results[0][7]))
        filter=Filter(filter[0][1], filter[0][0])
    except:
        pump=None
        filter=None

    pondDepth=results[0][1]
    pondWidth=results[0][3]
    pondLength=results[0][2]
    waterTemp=results[0][4]
    waterLevel=results[0][5]
    pondID=results[0][0]
    days=results[0][8]

    pond=Pond(width=pondWidth, length=pondLength, depth=pondDepth, temp=waterTemp, waterLevel=waterLevel, pump=pump, filter=filter, pondID=pondID, days=days) #creating pond from data
    return pond

#deleted a pond for the pond table
def deletePond(ID):
    if(ID == None):
        correct=False
        results=pondSimDatabase.getPonds(db,cursor)
        print('')
        print('|{0:^10}|{1:^10}|{2:^15}|{3:^15}|{4:^15}|{5:^15}|{6:^10}|{7:^10}|'.format('PondID','pondDepth','PondLength','PondWidth','WaterTemp','waterLevel','pump','filter'))
        for each in range(0,len(results)):
            print('{0:^12} {1:^10} {2:^10} {3:^20} {4:^10} {5:^20} {6:^5} {7:^15}'.format(results[each][0],results[each][1],results[each][2],results[each][3],results[each][4],results[each][5],results[each][6],results[each][7]))
        print('')
        #getting ID from user
        try:
            while correct==False:
                IDchoice=int(input('Enter the ID of the pond you wish to delete: '))
                if IDchoice>len(results)or IDchoice<0:
                    print("That ID was not found on the list")
                else:
                    correct=True
        except:
            print("That was not a valid ID - please enter a number")
        filter=pond.getFilter()
        filterID=filter.getFilterID()
        pump=pond.getPump()
        pumpID=pump.getPumpID()
        pondSimDatabase.deletePond(db, cursor, IDchoice)
        pondSimDatabase.deleteFishFromPond(db, cursor, IDchoice)
        pondSimDatabase.deletePlantFromPond(db,cursor,IDchoice)
        pondSimDatabase.deletePump(db,cursor,pumpID)
        pondSimDatabase.deleteFilter(db,cursor,filterID)
    else:
        pondSimDatabase.deletePond(db, cursor, ID)

#increment day
def incrementDay(mainPond,db,cursor):
    def updateFish(tup):
        sql = """update fish
                set hunger=?, daysAlive=?
                where fishID=?"""
        cursor.execute(sql,tup)
        db.commit()

    def updatePlants(tup):
        sql="""update plant
            set daysAlive=?, growth=?
            where plantID=?"""
        cursor.execute(sql,tup)
        db.commit()

    fishList=pondSimDatabase.fetchFishInfo(db,cursor,mainPond.getPondID())
    plantList=pondSimDatabase.fetchPlantInfo(db,cursor,mainPond.getPondID())
    plantType=pondSimDatabase.fetchPlantType(db,cursor)
    foodNeed=pondSimDatabase.fetchFoodNeed(db,cursor)

    for count in range(0,len(fishList)):
        daysAlive=fishList[count][4]+1
        if fishList[count][6]=="Goldfish":
            hunger=fishList[count][2]+foodNeed[0][1]
        elif fishList[count][6]=="Tench":
            hunger=fishList[count][2]+foodNeed[1][1]
        elif fishList[count][6]=="Orfe":
            hunger=fishList[count][2]+foodNeed[2][1]
        elif fishList[count][6]=="Koi":
            hunger=fishList[count][2]+foodNeed[3][1]
        elif fishList[count][6]=="Rudd":
            hunger=fishList[count][2]+foodNeed[4][1]
        elif fishList[count][6]=="Shark":
            hunger=fishList[count][2]+foodNeed[5][1]
        if fishList[count][3]=="Dead":
            hunger=0
            daysAlive=0
        elif fishList[count][3]=="good":
            daysAlive=fishList[count][4]+1
        tup=(hunger,daysAlive,fishList[count][0])
        updateFish(tup)

    for count in range(0,len(plantList)):
        daysAlive=plantList[count][4]+1
        if plantList[count][1]=="Rooted Floater":
            growth=plantList[count][5]+plantType[0][1]
        elif plantList[count][1]=="True Floater":
            growth=plantList[count][5]+plantType[1][1]
        elif plantList[count][1]=="Emergent":
            growth=plantList[count][5]+plantType[2][1]
        elif plantList[count][1]=="submersed":
            growth=plantList[count][5]+plantType[3][1]
        if plantList[count][2]=="Dead":
            daysAlive=0
        else:
            daysAlive=plantList[count][4]+1
        tup=(daysAlive,plantList[count][5]+1,plantList[count][0])
        updatePlants(tup)

    randomNo=random.randint(0,20)
    if randomNo==1:
       try:
           pump=pondSimDatabase.fetchPump(db,cursor,mainPond.getPump()) #fetching pump data
           pump=Pump(pump[0][1], pump[0][0]) #creating pump object from pump data
           pumpStatus=pump.getPumpStatus()
           pump.setPumpStatus("Broken")
           pump.updatePump(db,cursor)
           mainPond.setPump(pump.getPumpID())
       except:
            pass



    randomNo=random.randint(0,20)
    if randomNo==1:
        try:
            filter=pondSimDatabase.fetchFilter(db,cursor,self.mainPond.getFilter()) #fetching pump data
            filter=Filter(filter[0][1], filter[0][0]) #creating pump object from pump data
            filterStatus=filter.getFilterStatus()
            filter.setFilterStatus("Broken")
            filter.updateFilter(db,cursor)
            mainPond.setFilter(filter.getFilterID())
        except:
            pass


    mainPond.setDays(int(mainPond.getDays())+1)
    mainPond.updatePond(db,cursor)

#manage pond menu - displays when a pond is opened or created
def managePond(pond, db, cursor):
    def displayMenu():
        print('')
        print("1. Check pond status")
        print("2. Manage pump and filter")
        print("3. Edit pond settings")
        print("4. Manage pond life")
        print("5. Increment Day")
        print("6. Exit to main menu")
        print('')

    def getMenuChoice():
        displayMenu()
        choice=0
        while choice==0:
            try:
                choice=int(input("What would you like to do? "))
                if choice>6 or choice<1:
                    print("You must enter an option between 1 and 6")
            except:
                print("You must enter a number between 1 and 6")
        return choice

    def showPondStatus():
        #displaying pond information
        print('')
        print('Day',pond.getDays())
        print('')
        print("Pond depth:",pond.getPondDepth())
        print("Pond Length:", pond.getPondLength())
        print("Pond Width:", pond.getPondWidth())
        print("Water Level:",pond.getWaterLevel())
        print("Water Temperature:",pond.getWaterTemp())
        print('')

        pump=pond.getPump()
        filter=pond.getFilter()
        pumpStatus=pump.getPumpStatus()
        filterStatus=filter.getFilterStatus()
        print("Filter:",filterStatus)
        print("Pump:",pumpStatus)
        print("")

        fishTup=pondSimDatabase.fetchFish(db, cursor, pond.getPondID())
        plantsTup=pondSimDatabase.fetchPlants(db, cursor, pond.getPondID())

        #displaying list of fish in pond
        print('|{0:^10}|{1:^10}|{2:^15}|{3:^15}|{4:^15}|{5:^15}|{6:^10}|'.format('FishID','fishName','Hunger','Status','daysAlive','growth','FishType'))
        for each in range(0,len(fishTup)):
            print('{0:^12} {1:^10} {2:^10} {3:^20} {4:^10} {5:^20} {6:^5}'.format(fishTup[each][0],fishTup[each][1],fishTup[each][2],fishTup[each][3],fishTup[each][4],fishTup[each][5],fishTup[each][6]))
        print('')
        #displaying lists of plants in the pond
        print('|{0:^10}|{1:^10}|{2:^15}|{3:^15}|{4:^15}|{5:^15}|'.format('PlantID','PlantType','Status','LightNeed','daysAlive','growth'))
        for each in range(0,len(plantsTup)):
            print('{0:^12} {1:^10} {2:^10} {3:^20} {4:^10} {5:^20}'.format(plantsTup[each][0],plantsTup[each][1],plantsTup[each][2],plantsTup[each][3],plantsTup[each][4],plantsTup[each][5]))
        print('')

    def managePumpAndFilter():
        def displayMenu():
            print('1. Add pump and filter')
            print('2. Remove pump and filter')
            print('3. Edit pump and filter')

        def getMenuChoice():
            displayMenu()
            choice=0
            while choice==0:
                try:
                    choice=int(input("What would you like to do? "))
                    if choice>3 or choice<1:
                        print("You must enter an option between 1 and 3")
                except:
                    print("You must enter a number between 1 and 3")
            return choice

        def addPaF():
            newFilter=Filter("good",None)
            newPump=Pump("good",None)
            newFilter.saveFilter()
            newPump.savePump()
            pond.setFilter(newFilter)
            pond.setPump(newPump)
            pond.updatePond()

        def removePaF():
            filter=pond.getFilter()
            filterID=filter.getFilterID()
            pump=pond.getPump()
            pumpID=pump.getPumpID()
            pondSimDatabase.deletePump(db,cursor,pumpID)
            pondSimDatabase.deleteFilter(db,cursor,filterID)
            pond.setFilter(None)
            pond.setPump(None)
            pond.updatePond()

        def editPaF():
            pump=pond.getPump()
            filter=pond.getFilter()
            pumpStatus=pump.getPumpStatus()
            filterStatus=filter.getFilterStatus()
            print("Filter:",filterStatus)
            print("Pump:",pumpStatus)
            if filterStatus=="Broken":
                choice=int(input("Would you like to fix the filter? (yes = 1, no = 0) "))
                if choice==1:
                    filter.setFilterStatus("good")
                else:
                    pass
            if pumpStatus=="Broken":
                choice=int(input("Would you like to fix the pump? (yes = 1, no = 0) "))
                if choice==1:
                    pump.setPumpStatus("good")
                else:
                    pass

        choice=getMenuChoice()
        if choice==1:
            addPaF()
        if choice==2:
            removePaF()
        if choice==3:
            editPaF()



def manualGrow(amount,pond,db,cursor):
    def updateFish(tup):
        sql = """update fish
                set hunger=?, growth=?, status=?
                where fishID=?"""
        cursor.execute(sql,tup)
        db.commit()

    hunger=pondSimDatabase.fetchFishInfo(db,cursor,pond.getPondID())

    for count in range(0,len(hunger)):
        amount2=amount
        amount=amount-hunger[count][2]
        hunger2=hunger[count][2]-amount2
        if hunger2<0:
            hunger2=0

        growthList=pondSimDatabase.fetchGrowthRate(db,cursor)
        if hunger2==0 and hunger[count][3]!="Dead":
            if hunger[count][6]=="Goldfish":
                growth=growthList[0][0]+hunger[count][5]
            elif hunger[count][6]=="Tench":
                growth=growthList[1][0]+hunger[count][5]
            elif hunger[count][6]=="Orfe":
                growth=growthList[2][0]+hunger[count][5]
            elif hunger[count][6]=="Koi":
                growth=growthList[3][0]+hunger[count][5]
            elif hunger[count][6]=="Rudd":
                growth=growthList[4][0]+hunger[count][5]
            elif hunger[count][6]=="Shark":
                growth=growthList[5][0]+hunger[count][5]
        else:
            growth=hunger[count][5]
        fishID=hunger[count][0]

        if hunger[count][2]>20:
            status="Dead"
            growth=hunger[count][5]
        else:
            status=hunger[count][3]
        #hunger2=0
        fishID=hunger[count][0]
        tup=(hunger2,growth,status,fishID)
        updateFish(tup)
    incrementDay(pond,db,cursor)


def automaticGrow(db,cursor,pond):
    #feeds fish the required food amount for 7 days.
    def updateFish(tup):
        sql = """update fish
                set hunger=?, growth=?, status=?
                where fishID=?"""
        cursor.execute(sql,tup)
        db.commit()

    fishList=pondSimDatabase.fetchFishInfo(db,cursor,pond.getPondID())
    growthList=pondSimDatabase.fetchGrowthRate(db,cursor)
    for count in range(0,7):
        for count in range(0,len(fishList)):
            if fishList[count][3]!="Dead":
                if fishList[count][6]=="Goldfish":
                    growth=growthList[0][0]+fishList[count][5]
                elif fishList[count][6]=="Tench":
                    growth=growthList[1][0]+fishList[count][5]
                elif fishList[count][6]=="Orfe":
                    growth=growthList[2][0]+fishList[count][5]
                elif fishList[count][6]=="Koi":
                    growth=growthList[3][0]+fishList[count][5]
                elif fishList[count][6]=="Rudd":
                    growth=growthList[4][0]+fishList[count][5]
                elif fishList[count][6]=="Shark":
                    growth=growthList[5][0]+fishList[count][5]
            else:
                growth=fishList[count][5]
            if fishList[count][2]>20:
                status="Dead"
                growth=fishList[count][5]
            else:
                status=fishList[count][3]
            hunger=0
            fishID=fishList[count][0]
            tup=(hunger,growth,status,fishID)
            updateFish(tup)
        incrementDay(pond,db,cursor)


#main program
if __name__ == "__main__":
    db=sqlite3.connect('pond.db') #connecting to database
    cursor=db.cursor()
    databaseCreation.createDatabases()
    fish.saveFishTypes(db,cursor)
    plant.savePlantTypes(db,cursor)

    Exit=False

    while Exit==False:
        choice=mainMenu()
        if choice==1:
            pond=createPond()
            managePond(pond, db, cursor)
        if choice==2:
            pond=openPond()
            managePond(pond, db, cursor)
        if choice==3:
            ID=None
            deletePond(ID)
        if choice==4:
            Exit=True
    cursor.close()
```

10.6 pondSimDatabase Module

```
import sqlite3

#database queries
def getPonds(db,cursor):
    sql="""select *
            from pond"""
    cursor.execute(sql)
    results=cursor.fetchall()
    return results

def fetchPond(db,cursor,IDchoice):
    IDchoice=(IDchoice,)
    sql="""select *
            from pond
            where pondID=?"""
    cursor.execute(sql,IDchoice)
    results=cursor.fetchall()
    return results

def fetchPump(db,cursor,ID):
    ID=(ID,)
    sql="""select *
            from pump
            where pumpID=?"""
    cursor.execute(sql,ID)
    pumpResults=cursor.fetchall()
    return pumpResults

def fetchFilter(db,cursor,ID):
    ID=(ID,)
    sql="""select *
            from filter
            where filterID=?"""
    cursor.execute(sql,ID)
    filterResults=cursor.fetchall()
    return filterResults

def fetchFish(db,cursor,ID):
    ID=(ID,)
    sql="""select *
        from fish
        where pondID=?"""
    cursor.execute(sql,ID)
    fishResults=cursor.fetchall()
    return fishResults

def fetchPlants(db,cursor,ID):
    ID=(ID,)
    sql="""select *
        from plant
        where pondID=?"""
    cursor.execute(sql,ID)
    plantResults=cursor.fetchall()
    return plantResults

def fetchPlantInfo(db,cursor,ID):
    ID=(ID,)
    sql="""select *
        from plant
        where pondID=?"""
    cursor.execute(sql,ID)
    plantResults=cursor.fetchall()
    return plantResults

def fetchPlantType(db,cursor):
    sql="""select *
        from plantType"""
    cursor.execute(sql)
    plantResults=cursor.fetchall()
    return plantResults

def fetchFishType(db,cursor):
    sql="""select *
        from fishType"""
    cursor.execute(sql)
    fishResults=cursor.fetchall()
    return fishResults

def fetchGrowthRate(db,cursor):
    sql="""select growthRate
        from fishType"""
    cursor.execute(sql)
    foodNeed=cursor.fetchall()
    return foodNeed

def fetchFoodNeed(db,cursor):
    sql="""select *
        from fishType"""
    cursor.execute(sql)
    foodNeed=cursor.fetchall()
    return foodNeed

def fetchFishInfo(db,cursor,ID):
    ID=(ID,)
    sql="""select *
        from fish
        where pondID=?"""
    cursor.execute(sql,ID)
    hunger=cursor.fetchall()
    return hunger

def deletePond(db,cursor,ID):
    ID=(ID,)
    sql="""delete
        from pond
        where pondID=?"""
    cursor.execute(sql,ID)
    db.commit()
```

```

def fetchPondID(db,cursor):
    sql="""select pondID
        from pond"""
    cursor.execute(sql)
    IDList=cursor.fetchall()
    return IDList

def deleteFish(db,cursor,ID):
    ID=(ID,)
    sql="""delete
        from fish
        where fishID=?"""
    cursor.execute(sql,ID)
    db.commit()

def deleteFishFromPond(db,cursor,ID):
    ID=(ID,)
    sql="""delete
        from fish
        where pondID=?"""
    cursor.execute(sql,ID)
    db.commit()

def deletePlantFromPond(db,cursor,ID):
    ID=(ID,)
    sql="""delete
        from plant
        where pondID=?"""
    cursor.execute(sql,ID)
    db.commit()

def deletePlant(db,cursor,ID):
    ID=(ID,)
    sql="""delete
        from plant
        where plantID=?"""
    cursor.execute(sql,ID)
    db.commit()

def deletePump(db,cursor,ID):
    ID=(ID,)
    sql="""delete
        from pump
        where pumpID=?"""
    cursor.execute(sql,ID)
    db.commit()

def deleteFilter(db,cursor,ID):
    ID=(ID,)
    sql="""delete
        from filter
        where filterID=?"""
    cursor.execute(sql,ID)
    db.commit()
```