Documentation

CCS222-18 Object-oriented Programming
OCTO: Object Counting Traffic Bot
Aldwin John Tapican
Marjolo Mabuti

GETTING STARTED

Requirements:

- *Python version 3.8+*
- *YOLOv3 or YOLOv4 with COCO dataset.*

To get started, clone the GitHub repository for OCTO.
https://github.com/aj-tap/OCTO

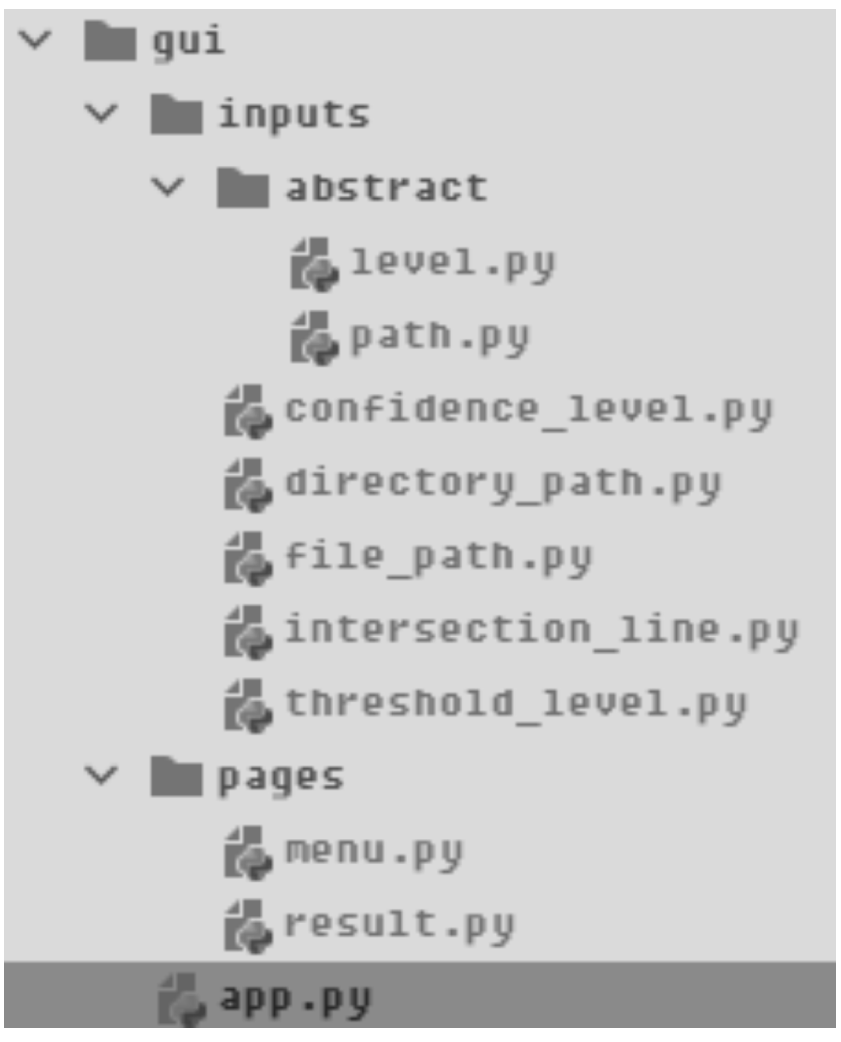
Before installing the libraries, it is recommended to first activate a Python virtual environment.

Navigate to the directory of the cloned repository and then run the following commands

```
pip install -r requirements.txt
pip install --upgrade imutils
pip install tk
```

You can now proceed to run main.py through your IDE or the terminal, and interact with the GUI.

I. OVERVIEW



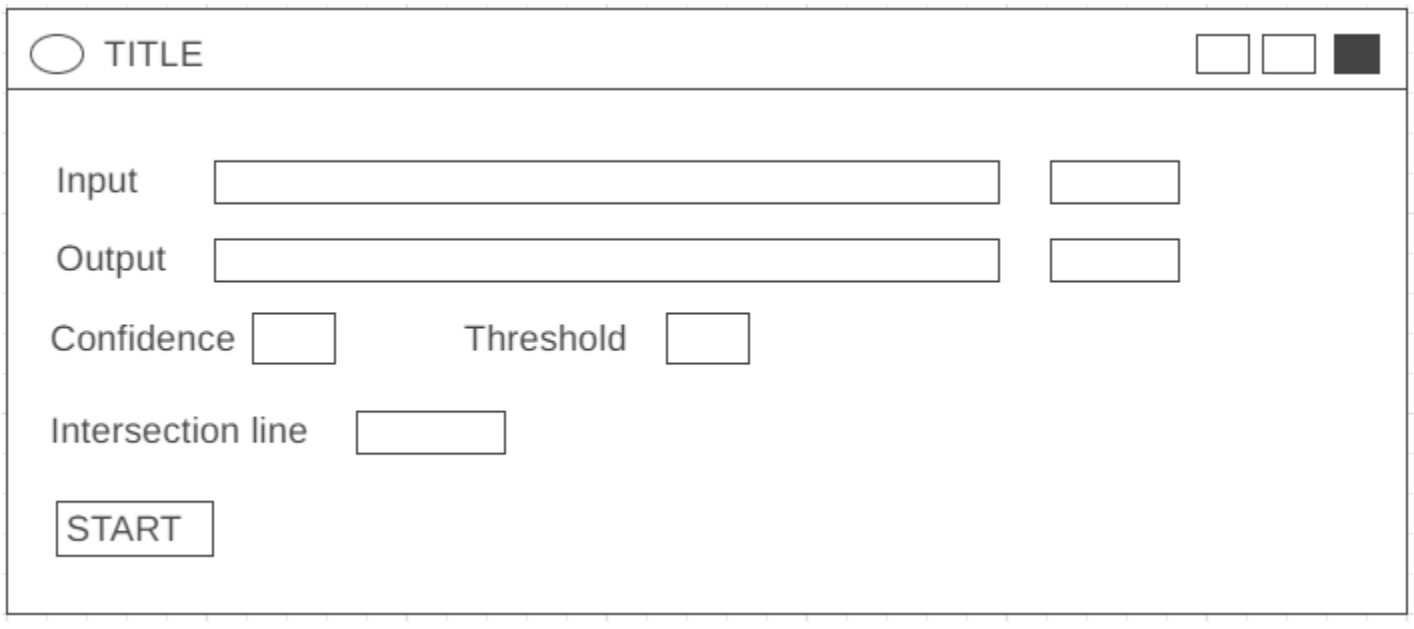
The main goals for writing the GUI of OCTO are simplicity, re-usability, and modularity. Using the Python library called tkinter we were able to create a simple GUI that works, and is able to fully interact with the back-end system by using the concepts in Object-oriented Programming. The Tk class represents the actual tkinter application or the window itself. We inherited Tk in the App class in order to define our own window configuration and to easily create logic for switching pages. We also used the Frame class from tkinter, mostly as a container to where we will render our widgets. The main frames being Menu and Result which are rendered inside the App class.

Our GUI works by loading Menu and Result into App, the App class is where both pages are managed. The Menu and Result classes will have a render_widgets() function that will render the relevant widgets to be used in their respective page. We created separate classes for each set of widgets to demonstrate the power of re-usability that we can unlock by using Object-oriented programming.

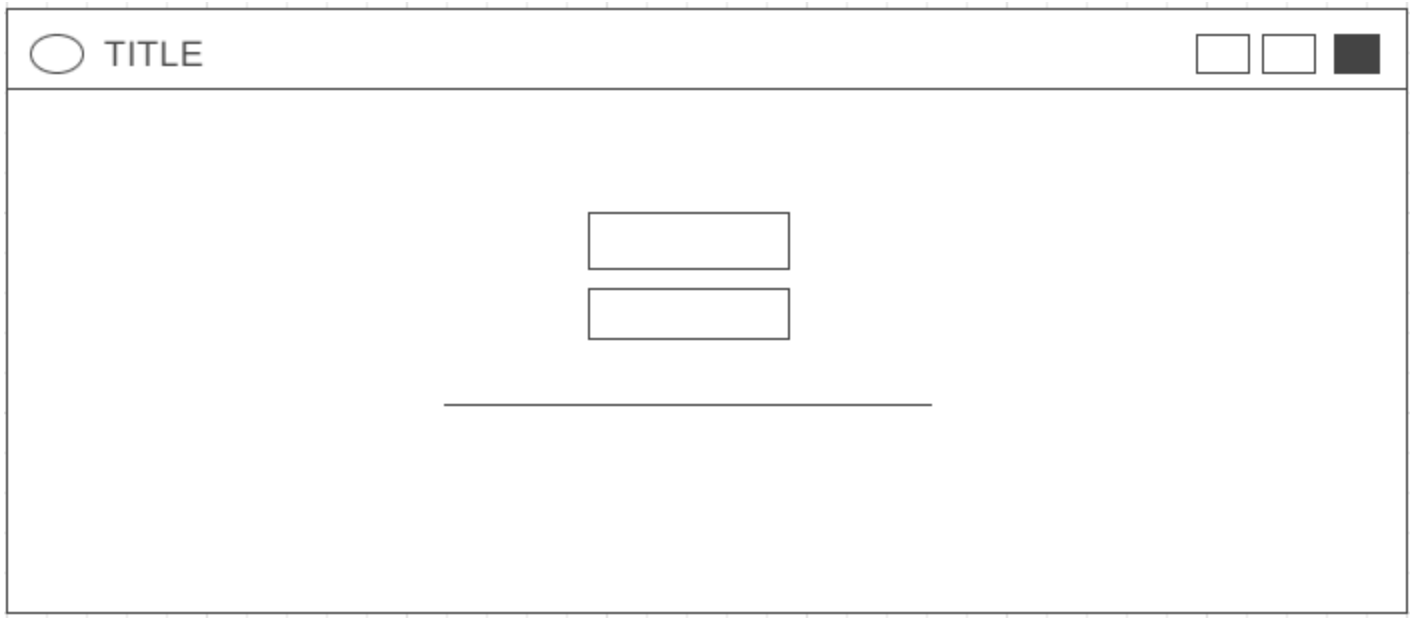
Each widget class has their own way of communicating with the back-end, made possible by OOP's feature of being able to access properties and methods from an outside class. This allowed our widgets to trigger setter methods defined in the back-end. The use of abstract classes has also been practiced in the front-end system to demonstrate how abstract classes can sometimes be useful in writing DRY code. We wrote the front-end system mostly with code modularity in mind.

II. WIREFRAME

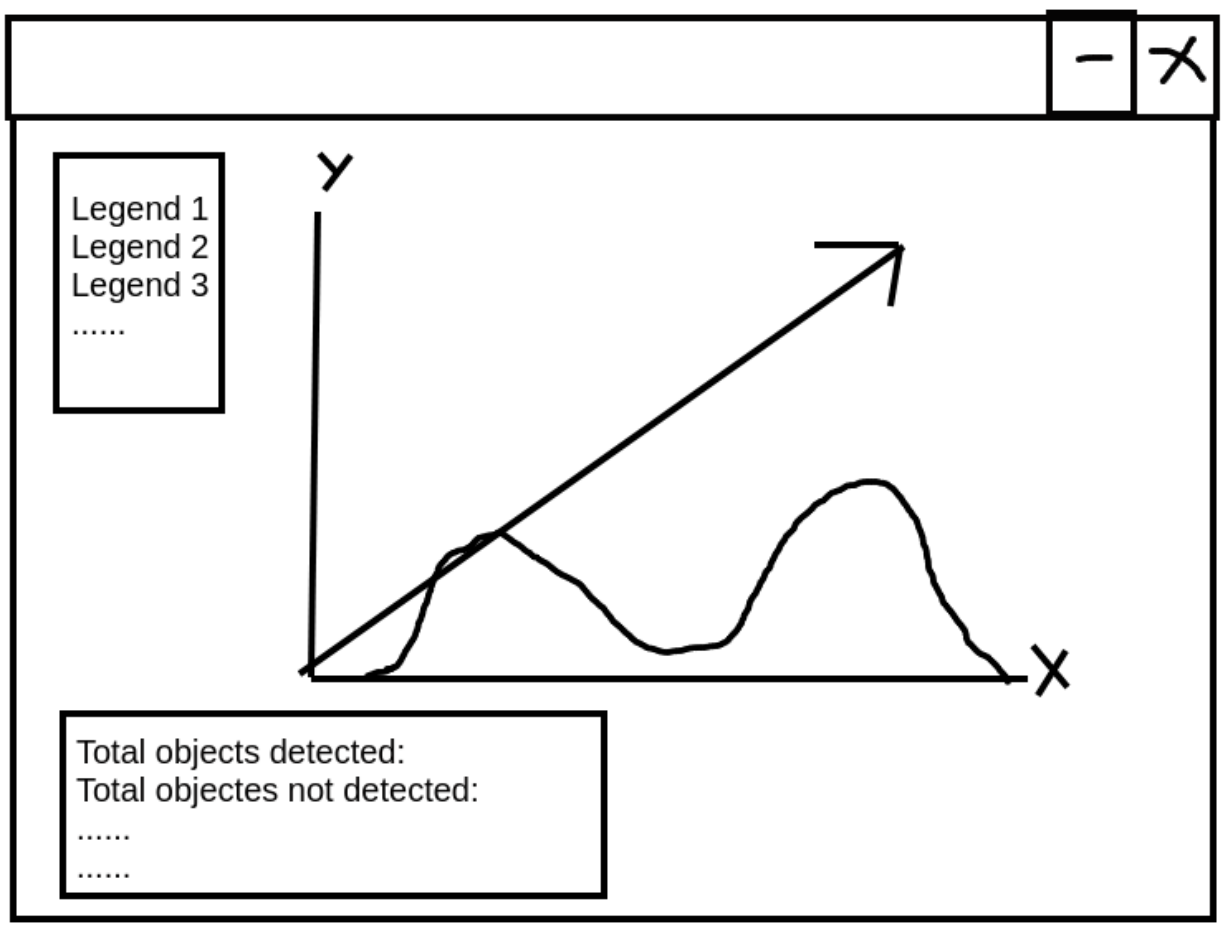
a. Menu page



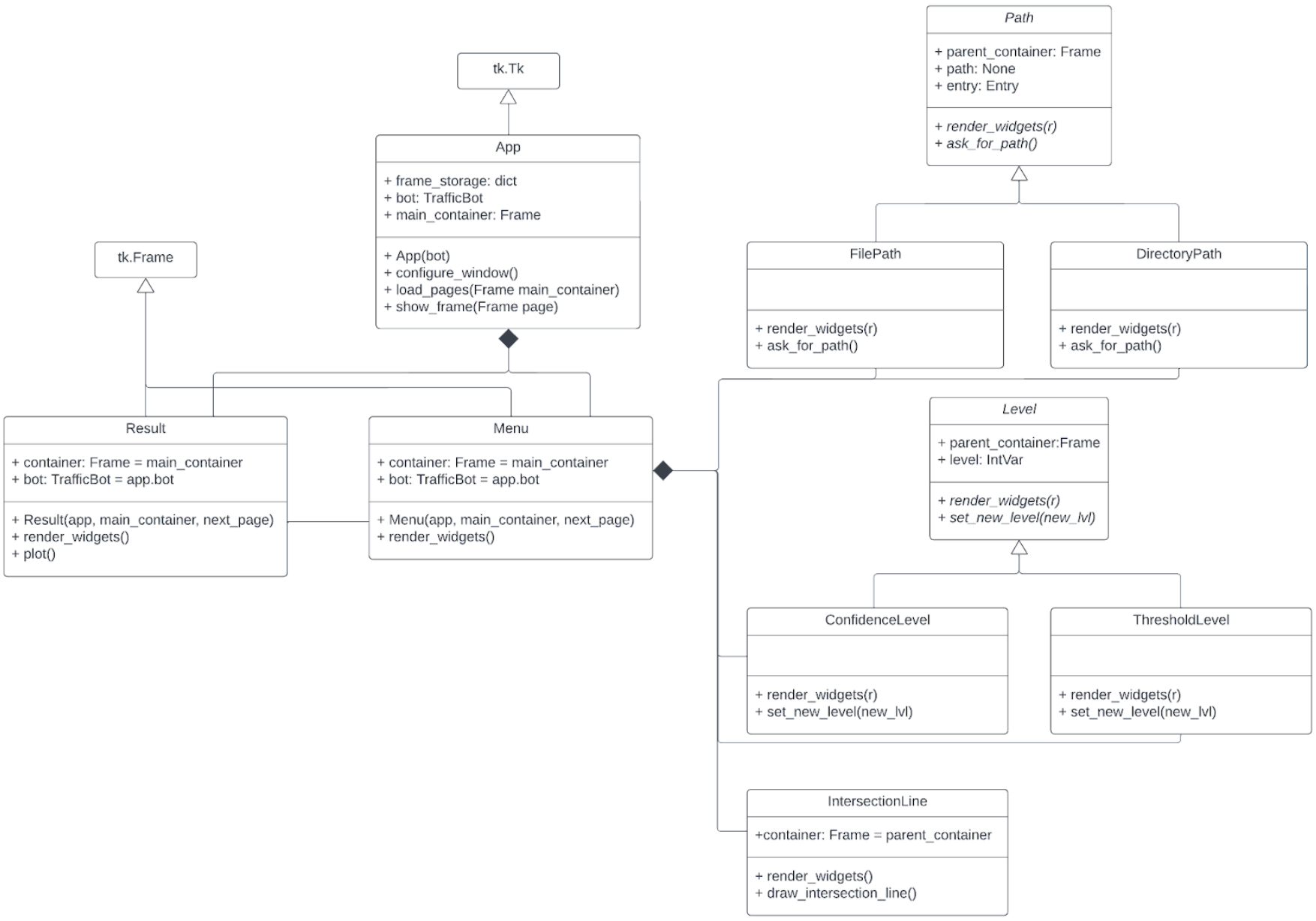
b. Result page



c. Result plot window



III. UML CLASS DIAGRAM



IV. CODE DOCUMENTATION

For the front-end system, we've decided that it's best to write the documentation within the source code to easily convey the purpose of each class and their parameters, and how each class can be connected. Code Documentation or writing documentation in the source code is widely used in the open-source community, because as programmers, code tells us how something works, and comments will tell us why. The process of documenting the code was made easier because we wrote our classes and functions with atomicity in mind.

app.py

```
from tkinter import Tk, Frame
from gui.pages.menu import Menu
from gui.pages.result import Result
```

```
class App(Tk):
    """
    A class to create the instance of the application itself. Inherits from the Tk object
    from the tkinter library in order to access some of its key methods and attributes.
    To create and configure our window.

    This entire class is built to interact with the bot instance and fully relies on its
    functions and attributes.

    ...
    Methods
    -------
    configure_window()
        Sets the icon and the title for the main window of the application.
    load_pages(main_container)
        Renders the frames unto the main_container in the form of a stack.
    show_frame(page)
        Raises the specified frame (or page) from the stack.
    """
```

```
def __init__(self, bot):
    """
    Parameters
    -------
    bot : bot
        The object counter / traffic bot instance. For the frames (or pages)
         to access the getters/setters of the bot (used for threshold, confidence,
         path, intersection line).
    """

    super().__init__()
    self.configure_window()

    self.frame_storage = {}
    self.bot = bot
    self.main_container = Frame(self)

    self.main_container.grid(row=0, column=0, padx=15, pady=15)
    self.load_pages(self.main_container)

def configure_window(self):
    self.resizable(False, False)
    self.wm_iconbitmap('../assets/traffic.ico')
    self.title('OCTO bot')

def load_pages(self, main_container):
    """
    Parameters
    -------
    main_container: Frame
        This is the container where the Menu and Result frames (or pages) will be rendered into.
    """

    self.frame_storage[Menu] = Menu(self, main_container, Result)
    self.frame_storage[Result] = Result(self, main_container, Menu)
    self.show_frame(Menu)

def show_frame(self, page):
    """
    Parameters
    -------
    page: Any
        The specified page is the frame to be raised or shown.
    """

    frame = self.frame_storage[page]
    frame.tkraise()
```

---

menu.py

```
from tkinter import Frame
from tkinter.ttk import Button

from gui.inputs.confidence_level import ConfidenceLevel
from gui.inputs.directory_path import DirectoryPath
from gui.inputs.file_path import FilePath
from gui.inputs.intersection_line import IntersectionLine
from gui.inputs.threshold_level import ThresholdLevel


class Menu(Frame):
    """
    A class that inherits tkinter Frame properties where we
    will render our menu related tkinter widgets into.

    ...
    Methods
```

```
-------
render_widgets()
    Renders the rest of the necessary widgets needed for the
    application.
"""

def __init__(self, app, main_container, next_page):
    """
    Parameters
    -------
    app : App
        The App class itself, in order for us to access
        the properties of the bot which is also a property of App
        which we will then need to set the necessary data for
        the threshold, confidence paths, and intersection line.
    main_container : Frame
        This is where the Menu page will be rendered into.
    next_page : Any
        This is the next page to be raised, for our purposes
        we will raise the Result page.
    """

    super().__init__(main_container)
    self.bot = app.bot
    self.render_widgets()

    Button(self, text="Start",
           command=lambda: app.show_frame(next_page))\
        .grid(row=5, column=1, ipadx=15)

    self.grid(row=0, column=0, sticky='nsew')

def render_widgets(self):
    """
    Parameters
    --------
    self :
        refers to Menu itself, in which the widgets will be
        rendered into.
    """

    FilePath(self).render_widgets(1)
    DirectoryPath(self).render_widgets(2)
    ConfidenceLevel(self).render_widgets(3)
    ThresholdLevel(self).render_widgets(3)
    IntersectionLine(self).render_widgets(4)
```

---

result.py

```
from tkinter.ttk import Label, Button
from tkinter import Frame, Tk

from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg, NavigationToolbar2Tk)


class Result(Frame):
    """
    A class that inherits tkinter Frame properties where we
    will render our result related tkinter widgets into.
    """

    def __init__(self, app, main_container, previous_page):
        """
        Parameters
        -------
        app : App
            The App class itself, in order for us to access
```

```
    the properties of the bot which is also a property of App
    which we will then need to set the necessary data for
    the threshold, confidence paths, and intersection line.
main_container : Frame
    This is where the Menu page will be rendered into.
previous_page : Any
    This is to raise the previous page---Menu.

...
Note; self refers to the Result page itself, which is a frame, this means
we can render widgets into self.
"""

super().__init__(main_container)

Button(self, text="Plot", command=lambda: plot()).pack()

Button(self, text="Back",
       command=lambda: app.show_frame(previous_page)).pack()

Label(self, text="By: Aldwin Tapican and Marjolo Mabuti").pack()

self.grid(row=0, column=0, sticky='nsew')
```

---

level.py

```
from abc import ABC, abstractmethod
from tkinter import IntVar


class Level(ABC):
    """
    This is an abstract class for threshold
    and confidence to avoid repeating
    the same constructor.

    ...
    Methods
    -------
    render_widgets(r)
        Requires implementing classes to have a function
        that will render widgets.
    set_new_level(new_lvl)
        Requires implementing classes to have a function
        that will set the user's desired level.
    """

    def __init__(self, parent_container):
        """
        Parameter
        -------
        parent_container : Frame
            Where we will render our widgets into. Also, so we can access
            the bot property.
        """

        self.parent_container = parent_container
        self.level = IntVar()
        self.level.set(50)

    @abstractmethod
    def render_widgets(self, r):
        pass

    @abstractmethod
    def set_new_level(self, new_lvl):
        pass
```

---

confidence_level.py

```
from tkinter.ttk import Label, OptionMenu

from gui.inputs.abstract.level import Level


class ConfidenceLevel(Level):
    """
    This class is for widgets and functionalities
    that are related to the input of the minimum confidence
    level.

    ...
    Methods
    --------
    render_widgets(r)
        Renders the widgets related to the input of minimum confidence
        level such as the OptionMenu.
    set_new_level(new_lvl)
        Accesses the bot property from parent_container to use the
        setter method defined in the backend to set the desired
        minimum confidence level.
    """

    def __init__(self, parent_container):
        """
        Parameter
        -------
        parent_container : Frame
            Where we will render our widgets into. Also, so we can access
            the bot property.
        """

        super().__init__(parent_container)

    def render_widgets(self, r):
        """
        Parameter
        -------
        r : int
            An integer to specify the row where we will render the widget.
        """

        Label(self.parent_container, text="Min confidence % : ") \
            .grid(sticky='w', row=r, column=1)

        OptionMenu(self.parent_container, self.level,
                   command=self.set_new_level,
                   *range(50, 101, 10))\
            .grid(sticky='w', row=r, column=2)

    def set_new_level(self, new_lvl):
        """
        Parameter
        -------
        new_lvl : int
            An integer to specify the desired minimum confidence level.
        """

        self.parent_container.bot.setConfidence(new_lvl * .01)
```

---

threshold_level.py

```
from tkinter.ttk import Label, OptionMenu

from gui.inputs.abstract.level import Level
```

```
class ThresholdLevel(Level):
    """
    This class is for widgets and functionalities
    that are related to the input of the threshold level.

    ...
    Methods
    --------
    render_widgets(r)
        Renders the widgets related to the input of threshold level
        such as the OptionMenu.
    set_new_level(new_lvl)
        Accesses the bot property from parent_container to use the
        setter method defined in the backend for the threshold level.
    """

    def __init__(self, parent_container):
        """
        Parameter
        -------
        parent_container : Frame
            Where we will render our widgets into. Also, so we can access
            the bot property.
        """

        super().__init__(parent_container)

    def render_widgets(self, r):
        """
        Parameter
        -------
        r : int
            An integer to specify the row where we will render the widget.
        """

        Label(self.parent_container, text="Threshold level % : ") \
            .grid(sticky='e', row=r, column=3)

        OptionMenu(self.parent_container, self.level,
                   command=self.set_new_level,
                   *range(40, 101, 10)) \
            .grid(sticky='w', row=r, column=4)

    def set_new_level(self, new_lvl):
        """
        Parameter
        -------
        new_lvl : int
            An integer to specify the desired threshold level.
        """

        self.parent_container.bot.setThreshold(new_lvl * .01)
```

---

path.py

```
from abc import ABC, abstractmethod
from tkinter.ttk import Entry


class Path(ABC):
    """
    This is an abstract class for file_path
    and directory_path to avoid repeating
    the same constructor.

    ...
    Methods
    -------
```

```
render_widgets(r)
    Requires implementing classes to have a function
    that will render widgets.
ask_for_path()
    Requires implementing classes to have a function
    that will input a path.
"""

def __init__(self, parent_container):
    """
    Parameter
    -------
    parent_container : Frame
        Where we will render our widgets into. Also, so we can access
        the bot property.
    """

    self.parent_container = parent_container
    self.path = None
    self.entry = Entry(parent_container)

@abstractmethod
def render_widgets(self, r):
    pass

@abstractmethod
def ask_for_path(self):
    pass
```

---

file_path.py

```
from tkinter import filedialog as fd
from tkinter.ttk import Label, Button

from gui.inputs.abstract.path import Path


class DirectoryPath(Path):
    """
      A class that renders the widgets that are related to accepting
      the path of the directory selected by the user.

      ...
      Methods
      -------
      render_widgets(r)
          Renders the widgets related to the input of the directory path.
      ask_for_path()
          Will open a filedialog for the user to select the desired
          directory in which the output of the program will be saved.
          The saved path will be of type string.

      ...
      Note; The setter for the video input file path of the bot
      is accessed through the parent_container.
      """

def __init__(self, parent_container):
    """
    Parameter
    -------
    parent_container : Frame
        Where we will render our widgets into. Also, so we can access
        the bot property.
    """

    super().__init__(parent_container)
```

```
def render_widgets(self, r):
    """
    Parameter
    -------
    r : int
        An integer to specify the row where we will render the widget.
    """

    Label(self.parent_container, text="Output directory: ") \
        .grid(sticky='w', row=r, column=1)

    self.entry.grid(sticky='w', ipadx=80, row=r, column=2, columnspan=3)

    Button(self.parent_container, text="Choose directory", command=lambda: self.ask_for_path()) \
        .grid(sticky='w', row=r, column=5)

def ask_for_path(self):
    self.path = fd.askdirectory()
    self.entry.insert(0, self.path)

    self.parent_container.bot.setOutput(self.path)
```

---

directory_path.py

```
from tkinter import filedialog as fd
from tkinter.ttk import Label, Button

from gui.inputs.abstract.path import Path


class DirectoryPath(Path):
    """
      A class that renders the widgets that are related to accepting
      the path of the directory selected by the user.

      ...
      Methods
      -------
      render_widgets(r)
          Renders the widgets related to the input of the directory path.
      ask_for_path()
          Will open a filedialog for the user to select the desired
          directory in which the output of the program will be saved.
          The saved path will be of type string.

      ...
      Note; The setter for the video input file path of the bot
      is accessed through the parent_container.
      """

    def __init__(self, parent_container):
        """
        Parameter
        -------
        parent_container : Frame
            Where we will render our widgets into. Also, so we can access
            the bot property.
        """

        super().__init__(parent_container)

    def render_widgets(self, r):
        """
        Parameter
        -------
        r : int
            An integer to specify the row where we will render the widget.
```

```
        """

        Label(self.parent_container, text="Output directory: ") \
            .grid(sticky='w', row=r, column=1)

        self.entry.grid(sticky='w', ipadx=80, row=r, column=2, columnspan=3)

        Button(self.parent_container, text="Choose directory", command=lambda: self.ask_for_path()) \
            .grid(sticky='w', row=r, column=5)

    def ask_for_path(self):
        self.path = fd.askdirectory()
        self.entry.insert(0, self.path)

        self.parent_container.bot.setOutput(self.path)
```

---

intersection_line.py

```
from tkinter.ttk import Label, Button

class IntersectionLine:
    """
    A class to render widgets related to the
    input of the intersection line.

    ...
    Methods
    -------
    render_widgets(r)
        Renders the widgets related to the intersection
        line
    draw_intersection_line()
        Accesses the bot property from the parent_container
        in order to use the trigger function to summon the
        openCV window that will accept two points to draw
        a line.
    """

    def __init__(self, parent_container):
        """
        Parameter
        -------
        parent_container : Frame
            Where we will render our widgets into. Also, so we can access
            the bot property.
        """

        self.parent_container = parent_container

    def render_widgets(self, r):
        """
        Parameter
        -------
        r : int
            An integer to specify the row where we will render the widget.
        """

        Label(self.parent_container, text="Intersection Line: ") \
            .grid(sticky='w', row=r, column=1)

        Button(self.parent_container, text="Draw", command=self.draw_intersection_line) \
            .grid(sticky='w', row=r, column=2)

    def draw_intersection_line(self):
        self.parent_container.bot.temporaryTriggerForIntersectionLine()
```