SOFTWARE DESIGN SPECIFICATIONS FOR NEUROMATIC

Submitted by:

Connor Arnold

Henry Gridley

Griffin Knipe

Daniel Vosburg

Date:

08/05/18

1. Introduction

1.1 Purpose of the Document

This document describes the data structures, system architecture, and component design composing the Neuromatic neural network application. Neuromatic is designed to provide a visual medium for the construction and training of neural networks.

1.2 Scope of the Development Project

The purpose of this product is to provide a visual tool to design and test neural network configurations as an alternative to writing code by hand. This will be accomplished through a graphical user interface where the user can place neural network layers on a canvas. The user will be able to edit the configuration of neural network layers, and specify the data on which the network will be trained, to create a working model for future use.

1.3 Definitions, Acronyms, and Abbreviations

1.3.1 Definitions

Term	**Definition**
Activation Function	A function that converts an input signal to an output signal within a neuron.
Canvas	The workspace, presented by the application graphical user interface, where the user will implement a neural network design.
Feature	An individual property being observed in a neural network.
Generate neural network	Create a python script using Keras, which can be used to create a model.
Hidden Layer	A network component, whose inputs and outputs are not exposed beyond the network.
Input Layer	A network component, whose input is exposed outside of the network and output is not.
Keras	A high level neural network API written in Python.
Layer	A collection of artificial neurons, at a specific depth, within a neural network.

Model	Mathematical representation of the parameters within a neural network. Makes predictions on data.
Multiprocessing	A Python module, which enables the parallel operation of processes.
Neural Network	A classification system emulating the computational behavior of the human brain.
Neuron	A function contained in the neural network which is trained to output a specific value when given an input data feature.
Output Layer	A network component, whose output is exposed outside of the network and input is not.
Pandas	An open source Python library, which provides alternative data structures to simplify data processing.
Tkinter	A Python module used for creating graphical user interfaces.
Train neural network	Use training data to modify parameters within the network and create a model.
Training Data	A processed collection of data points for which the desired output is already known. This data will be used to train the neural network.

Table 1 - Definitions

1.3.2 Acronyms and Abbreviations

Acronym/Abbreviation	**Meaning**
API	Application Programming Interface
CPU	Central Processing Unit
CSV	Comma Separated Values
GB	Gigabyte
GUI	Graphical User Interface

JSON	JavaScript Object Notation
MB	Megabyte
NN	Neural Network
RAM	Random Access Memory
SDS	Software Development Specifications
SRS	Software Requirements Specifications

Table 2 - Acronyms and Abbreviations

1.4 References

[1] C. Arnold, H. Gridley, G. Knipe, D. Vosburg. "Software Requirements Specifications for Neuromatic," unpublished.

[2] Docs.python.org. (2018). *24.1. Tkinter — Python interface to Tcl/Tk — Python 2.7.15 documentation*. [online] Available at: https://docs.python.org/2/library/tkinter.html [Accessed 19 Jul. 2018].

[3] Stack Overflow. (2018). *Stack Overflow - Where Developers Learn, Share, & Build Careers*. [online] Available at: https://stackoverflow.com/ [Accessed 19 Jul. 2018].

[4] TensorFlow. (2018). *API Documentation | TensorFlow*. [online] Available at: https://www.tensorflow.org/api_docs/ [Accessed 19 Jul. 2018].

1.5 Major Software Requirements

There are two main software requirements for this product, the first of which being TKinter. TKinter is Python's standard GUI package and is used to build the product's front end. This provides a visual wrapper around Keras, which is the second software requirement.

Keras is used to build, compile, and train the user's neural networks. The generated network scripts contain pre-formatted Keras object and API calls, and all of the network training is done natively by Keras.

1.6 Design Constraints, Limitations

1.6.1 Design Constraints

Training data must be in CSV format. Data sets cannot exceed a size of 500 MB. The product will be a desktop application compatible with Unix-based operating systems. The system running the application must have have at least 4 GB of RAM and at least a dual-core CPU faster than 2.0 GHz. The system constraints will affect the user at runtime and may limit the NN size, or the amount of data that can be used to train the model. System processing power and network training time have an inverse relationship.

1.6.2 Limitations

The limitations of this product are represented by the possible canvas and layer property values that can be set by the user. The bounds on these limitations are listed in Table 3.

Property	**Limitations**
Canvas Name	**Type:** String **Min:** 1 character **Max:** 32 characters **Restrictions:** Characters must be alphanumeric, dash, underscore, or period
Slot Count	**Type:** Integer **Min:** 3 **Max:** 10 **Restrictions:** None
Layer Type	**Type:** List of string **Choices:** “Input Layer”, “Hidden Layer”, “Dropout Layer”, “Output Layer” **Restrictions:** None
Layer Size	**Type:** Integer **Min:** 1 **Max:** 1000 **Restrictions:** None
Layer Data Dimensions	**Type:** Integer List - Length 2 **Min:** 1 **Max:** 100 **Restrictions:** None

Activation Function	**Type:** String **Choices:** “softmax”, “elu”, “selu”, “softplus”, “softsign”, “tanh”, “sigmoid”, “hard_sigmoid”, “linear” **Restrictions:** None
Dropout Layer Percentage	**Type:** Float **Min:** 0 **Max:** 1 **Restrictions:** None.

Table 3 - Limitations

1.7 Changes to Requirements

1.7.1 File Path Specification

- Original Requirement. SRS Section 2.3.12 [1].
  - “The user will be able to specify the file location of the created model.” [p. 6]
- Change to Requirement
  - The user will specify a project directory. The generated network script name will be inherited from the “Canvas Name”, specified by the canvas properties.
- Rationale
  - This consolidates the files generated by the application to a single folder. Therefore, the application will introduce minimal clutter to the user system’s file system.

1.7.2 Training Data File Size Limit

- Original Requirement. SRS Section 3.3 [1].
  - “The training data for the product will be in CSV format, and will have a maximum file size limit of 30 MB.” [p. 16]
- Change to Requirement
  - The maximum file size limit will be raised to 500 MB.
- Rationale
  - The Keras API has been tested using a training data CSV of 371 MB on a laptop using an dual-core CPU and 8GB of RAM. The training successfully completes in 75 seconds. The file size limit will be raised to allow the user to use a greater amount of training data, which will make models more accurate.

1.7.3 Timeout Non Functional Requirement

- Original Requirement. SRS Section 3.4.1 [1].
  - “When the “train” button is pressed, the product will train the NN on the training data, or display an error message if… A timeout interval is exceeded between processing each data point, indicating the user’s machine does not contain powerful enough hardware to train the network.” [p. 16]

- Change to Requirement
    - This requirement will be removed entirely. No timeout will be employed in the application.
- Rationale
    - This requirement was made under the assumption that the Keras API would make the processing of each training data point visible to the application using the API. However, instead of accepting single data points, the Keras API accepts the whole training data set and returns a result to the data set. This makes the implementation of a timeout, as described in the SRS, impossible.

1.8 Overview of the Document

The remainder of this document is organized into four parts: Data Design, System Architecture Description, and Detailed Description of Components.

Section 2, Data Design, describes the classes that compose the system and the relationships between classes using a high level class diagram. The function and data stored for each class is explained using lower level class diagrams. This section also describes the data file dependencies that exist outside of the system.

Section 3, System Architecture Description, describes the components that compose the application system. A UML System Architecture Diagram is provided to depict the dependencies between components.

Section 4, Detailed Description of Components, elaborates on the component descriptions provided in Section 3 by describing each components name, location in the system, type, purpose, function, subordinate modules, dependencies, resources, processing, and internal data.

Section 5, Interface Design, presents changes to interfaces described in the SRS.

2. Data Design

2.1 Data Objects and Resultant Data Structures

The classes instantiated by the application are presented in Figure 1.

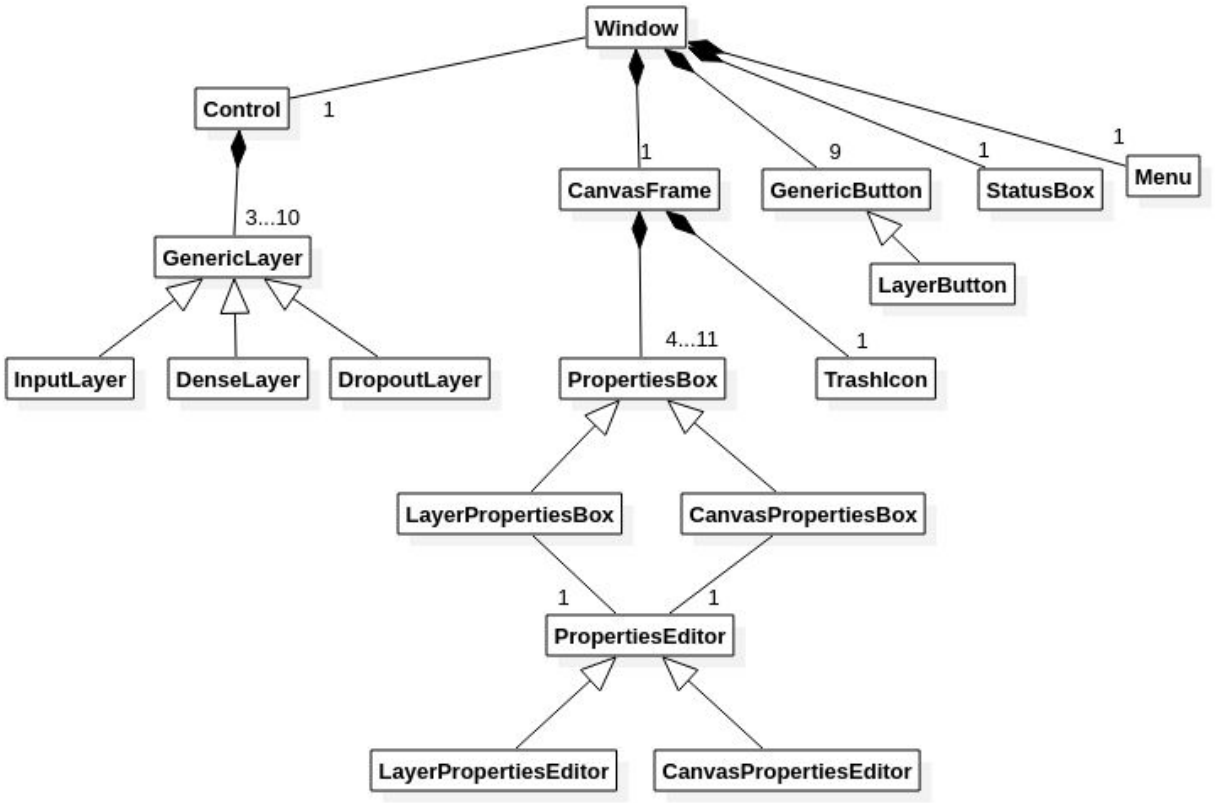

Figure 1 - Condensed Class Diagram showing relationships between classes.

The Window class is located at the highest level of the system. This class is responsible for organizing widgets, composing all classes except for the Control class, on the Tkinter window and interfacing those widgets with the Control class. More detailed class diagrams for the Window and Control classes are located in Appendices A and B, respectively.

The canvas design that is passed to the Control object is stored in the CanvasFrame object (Appendix D). The design data is stored in a list of "slots", in which each list element is a LayerPropertiesBox object (Appendix G), and in a CanvasPropertiesBox object (Appendix G). A LayerPropertiesBox object stores the size, name, and activation function of the layer. The CanvasPropertiesBox object stores the number of slots in a canvas design, the source file for the training data, the percentage split between training data and testing data, and the location to where the "Control" class will store its outputs. The parameters stored in the PropertiesBox

objects can be changed by the user using the dialog boxes created by the PropertiesEditor classes (Appendix F).

The LayerButton object (Appendix C) is used to determine the type of layers that exist on the canvas by passing its "layer_type" attribute a LayerPropertiesBox object after a drag and drop event. A LayerPropertiesBox object can be removed from the canvas design when the user triggers the TrashIcon object (Appendix E). This will trigger a command in the "CanvasFrame" that will remove the "LayerPropertiesBox" object from the slots list.

A Menu object (Appendix H) can save a CanvasFrame object as a JSON file. This JSON file can be reopened by the Menu object to reinitialize the saved canvas design.

A StatusBox object (Appendix I) displays a status string by using its "add_text" method. This method is passed to most GUI objects. Each GUI object stores this function as an attribute so that it can pass status strings to the function which are received by the StatusBox object.

Many GUI objects also store a root attribute and a "main_window" attribute. "Root" is a Tkinter concept which represents the widget within which the GUI object will be displayed. "main_window" is a reference to the Window object of the application. This allows a lower level GUI widget to pass data back to the highest level of the application without triggering a Window Tkinter event.

2.2 File and Database Structures

2.2.1 External File Structure

The user will be able to specify a directory where the generated network Python script and the trained model file are stored. The default directory will be a folder in the user's home directory. The names for the directory and files will be derived from the canvas name, specified by the user in the canvas properties.

2.2.2 Global Data

None

2.2.3 File and Data Cross Reference

The user must specify the location of the training data through the canvas properties window.

3. System Architecture Description

3.1 Overview of Modules/Components

This project contains 10 components which interact with one another to fulfill the requirements stated in the Neuromatic SRS.

Component Name	**Description**
Layer Properties Interface	This component records and displays the properties of each canvas layer to the user. This component also enables the user to manipulate those properties.
Layer Position Interface	This component records and displays the position of canvas layers and enables the user to add layers to the canvas and remove layers from the canvas.
Canvas Properties Interface	This component records and displays the properties of the canvas to the user and enables the user to manipulate those properties.
Status Interface	This component gathers the status of other components and displays the status to the user.
Canvas Open-Save Interface	This component enables the user to save the current canvas design, which includes each canvas layer's properties, the layer positions and the canvas properties. This saved state can be reinitialized by the application.
Generate Network Control	This component generates the network for the current canvas design.
Train Model Control	This component trains a network using a training data CSV file.
Stop Process Control	This component can terminate the Train Model Control component.

3.2 Structure and Relationships

The components described in section 3.1 are depicted in the system architecture diagram in Figure 2.

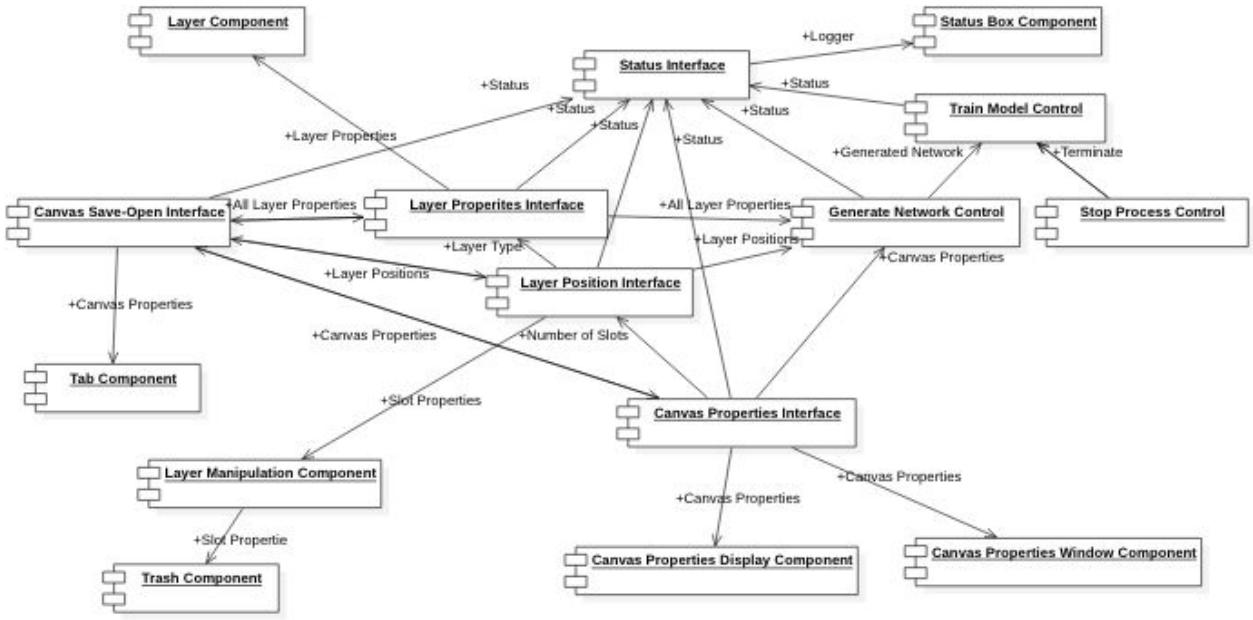

Figure 2 - System Architecture Diagram

The components can be split into two groups: Interfaces and Controls. Interfaces are components with which the user will be able to view and manipulate canvas design information. Controls are components which use information from Interfaces to generate application files.

The Layer Properties Interface depends on the Layer Position Interface to communicate the type of layer that has been placed in the canvas slot. The Layer Position Interface depends on the canvas properties interface to communicate the number of slots on the canvas. The Status Interface depends on all other components, except for the Stop Process Component, to communicate status updates.

The Canvas Open-Save Interface depends on the Layer Properties Interface to communicate the properties for each layer on the canvas, the Layer Position Interface to communicate where each layer exists on the canvas, and the Canvas Properties Interface to communicate the canvas properties. The Canvas Open-Save Interface will deliver the same information that it saved from each component to the component from which it was received. The Generate Network Control depends on the same information as the Canvas Open-Save Interface. The Train Model Control depends on the generated network to be communication by the Generate Network Control and the terminate signal from the Stop Process Control.

4. Detailed Description of Components

4.1 Layer Manipulation Component

Identification	Layer Manipulation Component: Part of User Interface. Component of Layer Properties interface and Layer Position Interface in Figure 2.
Type	Instances of class LayerPropertiesBox(PropertiesBox)
Purpose	This component represents the neural network layers defined by the user. Each NN layer is represented by a Tkinter Text Box acting as a slot. The user will drag and drop layer buttons from the right frame into a slot to update that slot's attributes. These slots will allow the user to edit the attributes of each layer in the neural network via a pop up box activated by a mouse click. These slots will hold all neural network information which will be used by the backend to generate a neural network. Fulfills SRS requirements 2.3.3 Add/Remove Layers and 2.3.4 Modify Component Properties
Function	● Drag and drop layer type ○ Input: Button click and drag from layer button ○ Output: Log Message ○ Result: Updated slot layer_type values ● Display slot attributes ○ Input: Layer_type attributes ○ Output: Displayed Values ○ Result: Show user the values of the designed network ● Edit slot properties ○ Input ■ Dropout ■ Activation ■ Size ○ Output: Displayed Value ○ Result: Updated layer attributes ● Remove slot attributes ○ Input: Drag and Drop slot box ○ Output: Default attributes ○ Result: Slot information is set to default values ● Return slot information ○ Input: Slot number ○ Output: Slot information ○ Result: Slot information sent to caller

Subordinates/Modules Used	● LayerPropertiesBox(PropertiesBox) ● LayerButton(GenericButton) ● Logger(object)
Dependencies	● Tkinter
Resources	● Screen/Display ● RAM ● CPU
Processing	Display slot attributes 1. Get current slot attributes 2. Update slot button text Drag and Drop Layer Type: 1. Button click and hold on layer button 2. Update main_window.layer_type to layer_button.layer_type 3. Click release on slot button to invoke slot.get_layer_type 4. Get main_window.layer_type 5. Update slot layer_type 6. Display slot attributes Edit slot properties 1. Button click on slot button 2. Slot properties pop up appears 3. Edit properties via dropdowns/editable text box 4. If cancel button pressed, close window, discard changes 5. If save button pressed, update slot attributes, close window 6. Display slot attributes Remove slot attributes 1. Slot button click and hold 2. Click release on trash icon 3. Slot button attributes set to default 4. Display slot attributes Returns slot information 1. Get slot attributes 2. Return slot attributes
Data	● layer_type ○ Data Type: String ○ Default Value: Empty ○ Description: Specifies what the what type of layer

	occupies the slot ● activation ○ Data Type: String ○ Default Value: sigmoid ○ Description: Specifies the activation function of the layer if layer type is input, hidden, or output. ● size ○ Data Type: Integer ○ Default Value: 1 ○ Description: Specifies the size of the layer if the layer type is input, hidden, or output. ● dropout ○ Data Type: Float ○ Default Value: .5 ○ Description: Specifies the dropout rate of the layer if the layer type is dropout ● slots ○ Data Type: List ○ Default Value: [] ○ Description: Holds the slots information for the layers currently on the canvas ● main_window ○ Data Type: Tkinter.tk ○ Default Value: None ○ Description: A reference to the main window where all canvas information can be found

4.2 Layer Component

Identification	Layer Component: Part of User Interface. Component of Layer Properties Interface in Figure 2.
Type	Instances of class LayerButton(GenericButton)
Purpose	This component will interact with the layer manipulation component to add layer types to the slots. This component will fulfill the SRS function requirement 2.3.3 Add/Remove Layer.
Function	● Dragged by user into slot to modify slot type ○ Input: User button click ○ Output: Invoke call to slot ○ Result
Subordinates/Modules	● LayerButton(GenericButton)

Used	● LayerPropertiesBox(PropertiesBox) ● Logger(object)
Dependencies	● Tkinter
Resources	● Screen/UI ● RAM ● CPU
Processing	Dragged by user into slot 1. On button click update main_window.layer_type 2. On motion drag to slot 3. On release find corresponding widget 4. If released on slot invoke slot.get_layer_type 5. Else do nothing
Data	● layer_type ○ Data Type: String ○ Default Value: None ○ Description: Holds the type of layer the button is associated with ● main_window ○ Data Type: Tkinter.tk ○ Default Value: None ○ Description: A reference to the main window where all canvas information can be found

4.3 Tab Component

Identification	Tab Component: Part of User Interface. Component of Canvas Open/Save in Figure 2
Type	Instance of class Tkinter.menu
Purpose	Allow the user to create a new blank canvas. Allows the user to open a previously saved canvas. Allow the user to save the current canvas. Fulfills SRS requirements 2.3.1 Open New Canvas, 2.3.7 Save Canvas, and 2.3.8 Load Canvas
Function	● New Canvas ○ Input: Button Click ○ Output: New canvas with def ○ Result: Set the window to its default settings ● Save Canvas ○ Input: Button Click

	○ Output: File write ○ Result: Canvas properties are saved to the user specified location ● Open Canvas ○ Input: ■ File location ■ Button Click ○ Output: ■ CanvasFrame ○ Result: ■ The file is read into the program and the properties are input to their canvas objects ● Help ○ Input: ■ Button click ■ search_string ○ Output: None ○ Result: The help bar will search for the user inputted string
Subordinates/Modules Used	● Tkinter.menu ● Window(object) ● CanvasFrame(object) ● Logger()
Dependencies	● Tkinter
Resources	● Memory ● RAM ● CPU
Processing	New Canvas 1. Click New from File dropdown. 2. 'Are you sure?' pop-up appears. 3. If yes reset all canvas and layer properties to default 4. Else return to previous window Save Canvas 1. Click Save from File dropdown 2. Get current save file path from canvas properties 3. Get current network design and canvas properties 4. Export data to specified file Open Canvas 1. Click Open from File dropdown 2. Select file from dropdown menu 3. If file is compatible import all canvas and slot data

	4. Else log error. Help 1. Click Help from upper tab 2. Search bar with options appears
Data	None

4.4 Status Box Component

Identification	Status Box Component: Part of User Interface. Component of Status Interface in Figure 2
Type	Instance of StatusBox
Purpose	Display log information to user from all other components. Fulfills SRS requirements 2.3.5 View Application Status
Function	● Log Information ○ Input: Message ○ Output: None ○ Result: Message is sent to the status box on the UI ● Scroll through logs ○ Input: Mouse scroll ○ Output: adjusted text ○ Result: The users scroll will move the text with the scroll direction
Subordinates/Modules Used	● tkinter.Frame() ● tkinter.Scrollbar() ● tkinter.Label() ● tkinter.Text() ● Logger()
Dependencies	Tkinter
Resources	● Ram ● Screen/Display
Processing	Log Information 1. Instantiate logger in root window 2. Pass logger to all components when instantiated 3. Receive new logs via the log() method 4. Output new messages to status box in bottom right Scroll through logs 1. If scroll is not at bottom, stay at current message,

	2. Else track latest messages 3. If scroll bar moves follow motion else stay at current location
Data	● Text ○ Data Type: String ○ Default: None ○ Description: The text to be outputted to the status box

4.5 Canvas Properties Display Component

Identification	Canvas Properties Display Component: Part of User Interface and Backend. Component of Canvas Properties Interface in Figure 2
Type	Instance of CanvasPropertiesBox(PropertiesBox)
Purpose	This module represents the current canvas properties that the user may edit. The user will be able to view the canvas name, the number of slots, and the filepath for the training data, Python script generation, and trained model through a text box on the canvas. The user will be able to edit these properties by double-clicking this text box, which will open a seperate window with fillable fields.
Function	● Display canvas properties ○ Input: Canvas properties ○ Output: None ○ Result: Canvas properties are displayed in their box ● Return canvas properties to other components ○ Input: None ○ Output: canvas property settings ○ Result: The canvas properties are returned to the caller
Subordinates/Modules Used	● tkinter.Button
Dependencies	● Tkinter
Resources	● Screen/Display, RAM
Processing	Display Canvas Properties

	1. Get canvas properties 2. Format properties in text box Return Canvas Properties 1. Get canvas properties 2. Return canvas properties in dictionary
Data	● canvas_name ○ Data Type: String ○ Default: new_canvas ○ Description: The save file name of the canvas properties ● project_directory ○ Data Type: String ○ Default: ~/Documents ○ Description: The path where files will be saved ● training_data ○ Data Type: String ○ Default: ~/Documents ○ Description: The path to the training data csv ● component_slots ○ Data Type: Integer ○ Default: 3 ○ Description: The number of slots displayed on the canvas ● training_size ○ Data Type: Float ○ Default: .8 ○ Description: The percentage split between training and testing data. If there was no split, the model could potentially memorize the data and be ineffective at classifying new data.

4.6 Canvas Properties Window Component

Identification	Canvas Properties Window Component: Part of User Interface and Backend. Component of Canvas Properties Interface in Figure 2
Type	Class
Purpose	This module represents the current canvas properties that can be edited. The user will be able to double-click the canvas properties text box and launch the window to edit the properties. The canvas

	name and slot counts will have an entry field to enter the information. The training data and project directory will have a "Browse" button that will launch a file browser, allowing the user to choose the file location. After editing these fields, the user will be able to save and exit this window by clicking an "ok" button, or the user can exit this window without saving the changes by clicking a "cancel" button.
Function	● Edit and update Canvas properties ○ Input: ■ Button Click ■ CanvasFrame ■ canvas_name ■ Training_data ■ project_directory ■ component_slots ■ training_size ○ Output: CanvasFrame ○ Result:The updated Canvas properties are saved to the CanvasFrame
Subordinates/Modules Used	● Logger() ● CanvasPropertiesBox(PropertiesBox)
Dependencies	● Tkinter
Resources	● Screen/Display ● RAM
Processing	Edit and Update Canvas Properties 1. Double click canvas properties 2. Canvas properties pop up appears 3. Edit properties via editable text box/file browser 4. If cancel button pressed, close window, discard changes 5. If save button pressed, update values 6. If component_slots changes add/remove slots from canvas 7. Remove pop up
Data	● canvas_name ○ Data Type: String ○ Default Value: New Canvas ○ Description: The name the file will be saved as in the project directory ● project_directory ○ Data Type: String

	○ Default Value: ~/Documents ○ Description: The path of the desired output directory ● component_slots ○ Data Type: Integer ○ Default Value: 3 ○ Description: The number of desired layers on the canvas. Min 2, Max 5 ● Training_size ○ Data Type: Float ○ Default Value: .8 ○ Description: The percentage split between training and testing data. If there was no split, the model could potentially memorize the data and be ineffective at classifying new data.

4.7 Trash Component

Identification	Trash Component: Part of User Interface. Component of Layer Properties Interface in Figure 2.
Type	Collection of Classes
Purpose	This module represents a destination for a layer to remove it from the canvas. The user will be able to drag a layer from a slot and place it in the trash component, which will change the slot into an empty slot.
Function	● Remove slot Attributes ○ Input: Button Release ○ Output: None ○ Result: The dragged slot will be set to empty with default settings
Subordinates/Modules Used	● LayerPropertiesBox(PropertiesBox)
Dependencies	● Tkinter
Resources	● RAM ● Screen/Display
Processing	Remove Slot Attributes 1. LayerProperyBox is clicked and dragged over trash component

	2. LayerPropertyBox acknowledges click release over trash component 3. LayerPropertyBox is set to Empty
Data	None

4.8 Generate Network Component

Identification	Generate Network Component: Part of Code Generation. Component of Generate Model Control in Figure 2.
Type	Control class
Purpose	This module satisfies SRS functional requirement Generate Python Script (2.3.9); the user will be able to generate the code for a neural network based on the components they have selected on the canvas.
Function	● Overview ○ Generate a script containing a neural network written using the Keras API ● Inputs ○ Types of layers ○ Output directory path ○ Optimizer type ○ Loss function ○ Metric types ○ Number of epochs ● Transformation ○ This module converts list of layers and their parameters to a neural network ● Outputs ○ Python file containing Keras neural network ● Results ○ Python script stored in user specified directory ○ Python script for importing stored in backend directory
Subordinates/Modules Used	● InputLayer(GenericLayer) ● DenseLayer(GenericLayer) ● DropoutLayer(GenericLayer)
Dependencies	● multiprocessing ○ Makes component thread safe ○ Limits sum of Generate Model processes and Train

	Model processes to 1
Resources	● Memory ● RAM ● CPU
Processing	Generate Script 1. Open “network.py” in user-specified directory 2. Write imports and function signature to file 3. For each layer a. Write line corresponding to layer to file 4. Write model compilation parameters to file 5. Write model fitness parameters to file 6. Write model saving lines to file 7. Copy file to backend directory 8. Set training flag to True
Data	● layers ○ Type: list of Layers ○ Default: [] ○ Description: All layers in the user-specified order ● canvas_properties ○ Type: dict of dicts ○ Default: {} ○ Description: User-specified properties used to correctly generate the neural network script. ● can_train ○ Type: boolean ○ Default: False ○ Description: Flag that determines if training can occur. Should only be set to True after a neural network script has been generated.

4.9 Train Model Component

Identification	Train Model Component: Part of Keras interface. Component of Train Model Control in Figure 2.
Type	Control module
Purpose	This module satisfies SRS functional requirement Train Model (2.3.13); the user will be able to train the neural network that has previously been generated.
Function	● Overview

	○ Train the Generated Network to create a NN ● Inputs ○ Training data CSV file ○ Generated neural network script (from 4.8) ● Transformation ○ Generated neural network accepts CSV training data ○ Training data is passed to Keras API, which creates a model ● Outputs ○ H5 file representing trained model ○ H5 file representing the weights in the model ○ JSON file representing the network architecture ● Results ○ Files stored in user specified directory
Subordinates/Modules Used	● UI generate button: The button clicked to begin training ● train_size: The percentage of data that is used for training. The percentage of data used for testing is 1-train_size.
Dependencies	● Keras ○ Neural network API ● pandas ○ Stores training data ● multiprocessing ○ Makes component thread safe ○ Limits sum of Generate Model processes and Train Model processes to 1 ● Generated network ○ Must be created by Generate Model Control (4.8) for this component to operate. ○ Accepts training data and trains the model
Resources	CPU execution time will have a direct relationship to the size of the neural network and the size of the training file. The greater the size, the longer execution will take.
Processing	1. If train button pushed a. Continue to (2) 2. If neural network exists: a. Continue to (4) 3. Else a. Raise NetworkException 4. Import neural network training function 5. Read CSV file into pandas DataFrame

	6. Split DataFrame into labels and features 7. Shuffle DataFrame and split into training and testing data 8. Run generated script
Data	● can_train ○ Type: boolean ○ Default: False ○ Description: Flag that determines if training can occur. Should only be set to True after a neural network script has been generated. ● X_train ○ Type: pandas DataFrame ○ Default: N/A ○ Description: Training features to be fed to the network ● y_train ○ Type: pandas DataFrame ○ Default: N/A ○ Description: Training labels to be fed to the network. The labels are read from the first column in the CSV file. ● X_test ○ Type: pandas DataFrame ○ Default: N/A ○ Description: Testing features to be fed to the network ● y_test ○ Type: pandas DataFrame ○ Default: N/A ○ Description: Testing labels to be fed to the network. The labels are read from the first column in the CSV file.

4.10 Stop Process Component

Identification	Stop Process Component: Part of Cancellation. Component of Stop Process Control in Figure 2
Type	Thread termination
Purpose	This module satisfies SRS functional requirement Stop Progress (2.3.10); the user will be able to cancel the training process using a UI button without crashing the program.

Function	● Overview ○ Cancel neural network training ● Inputs ○ None ● Transformation ○ N/A ● Outputs ○ None ● Results ○ Training is cancelled and control is returned to the user
Subordinates/Modules Used	● UI cancel button: The button clicked to terminate the process
Dependencies	● multiprocessing ○ Terminates network training thread
Resources	N/A
Processing	1. User clicks cancel button 2. Error message is logged 3. If thread lock is locked a. Release lock 4. Process is terminated
Data	None

5. Interface Design

The current GUI design has changed from the design presented in the SRS [1] Appendix C [p. 19]. These changes are the result of the functional requirement changes mentioned in Section 1.7.1 of this document and of the addition of a supported layer type.

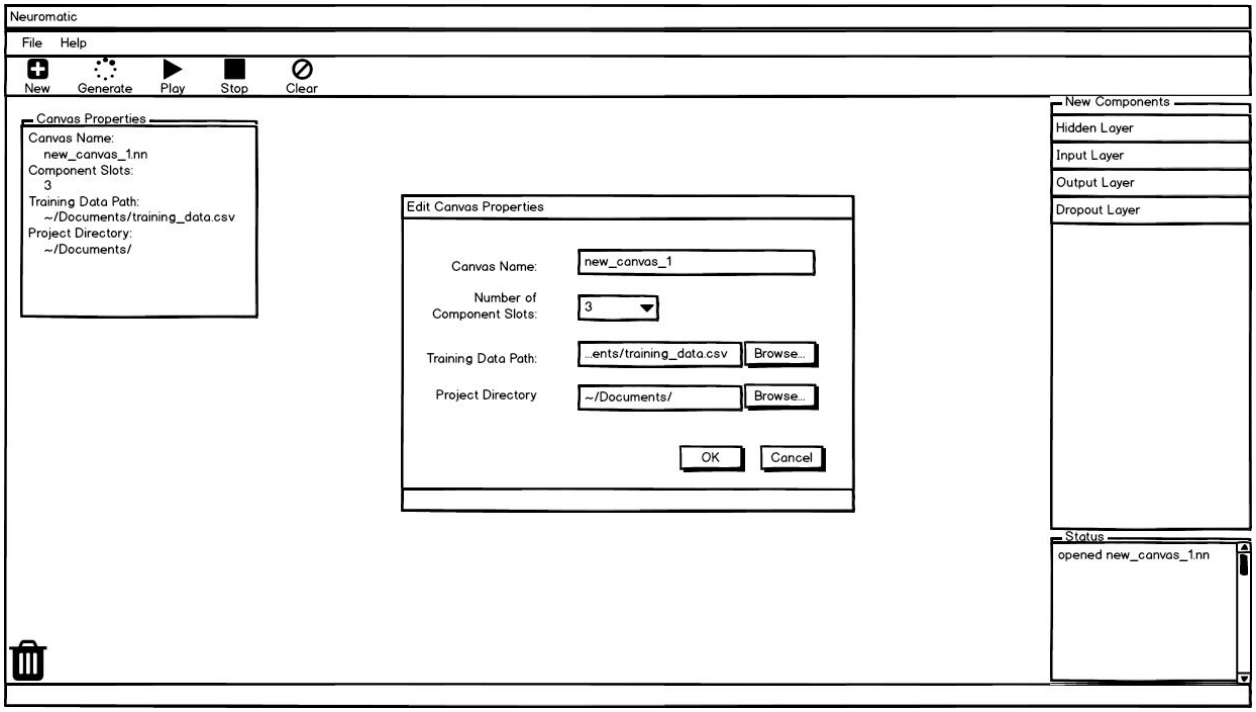

Figure. 3 - Wireframe representing interface changes from the SRS GUI design.

The user will no longer specify a path to the generated network script. Instead, the user will specify the project directory using the “Project Directory” field in Figure 3. The “Dropout Layer” has been added to the “New Components” menu on the right side of the screen.

APPENDIX

Appendix A

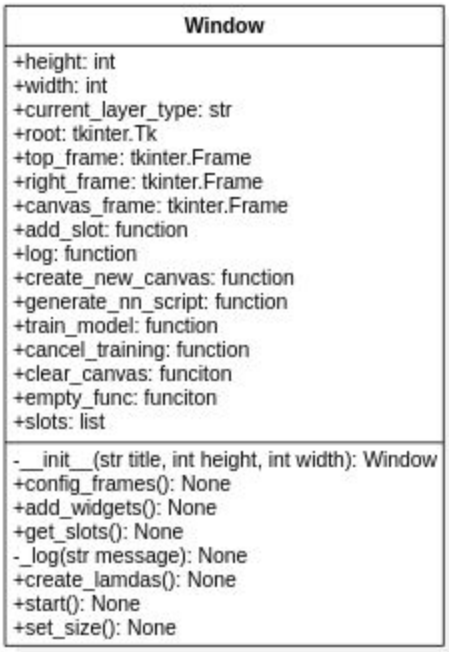

Figure 4 - Window class diagram

Appendix B

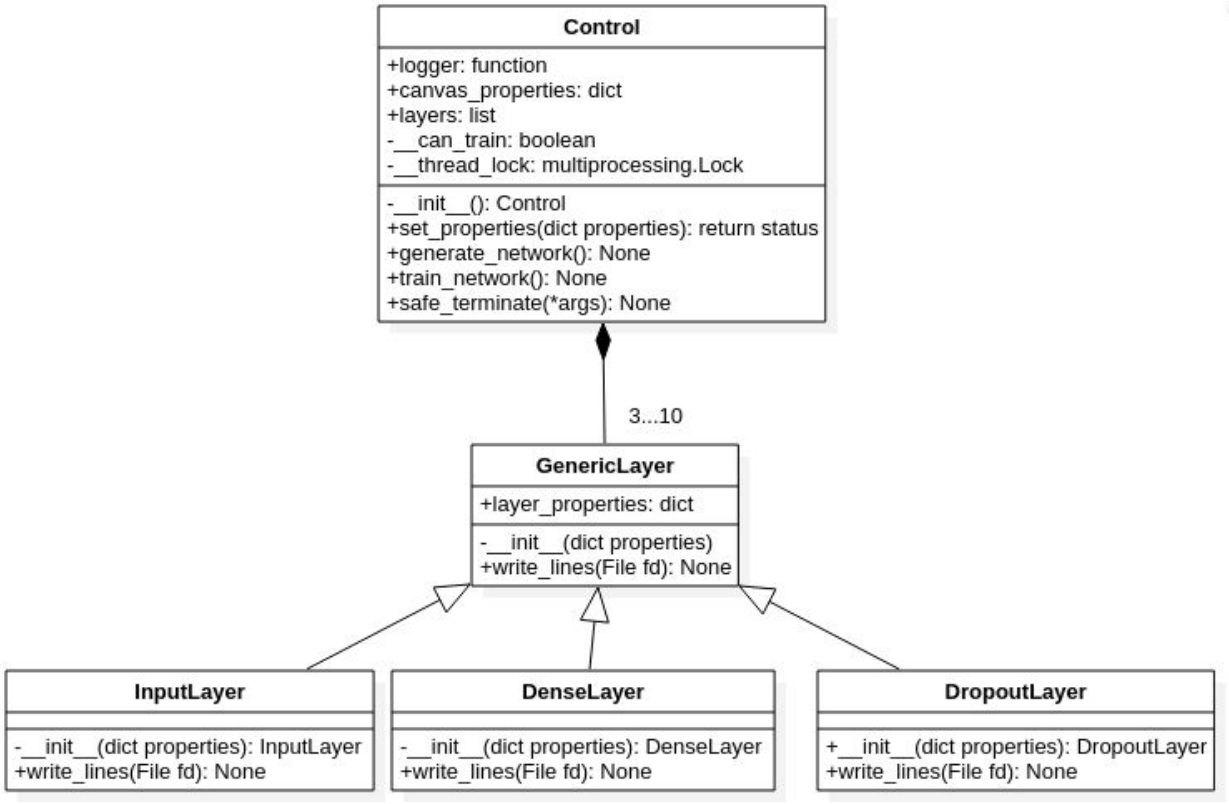

Figure 5 - Control and Layer class diagram

Appendix C

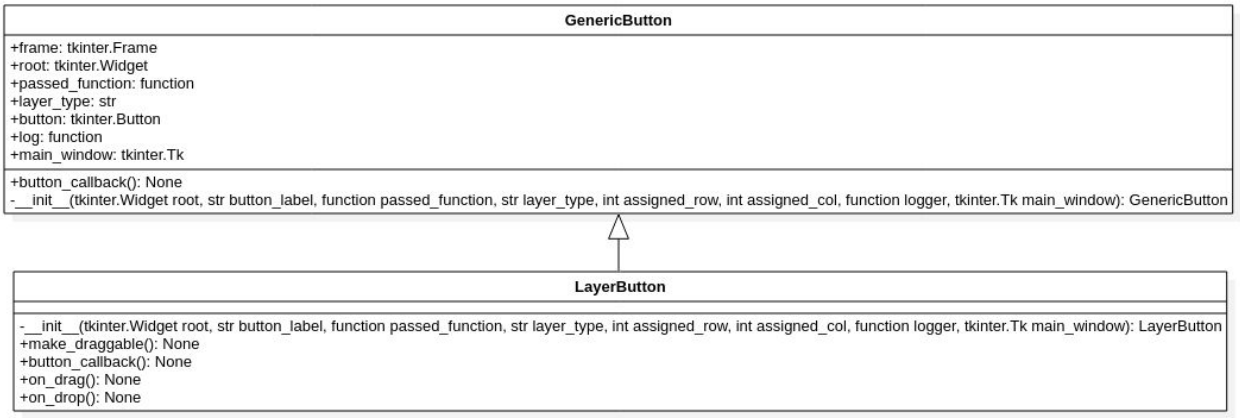

Figure 6 - Buttons class diagram

Appendix D

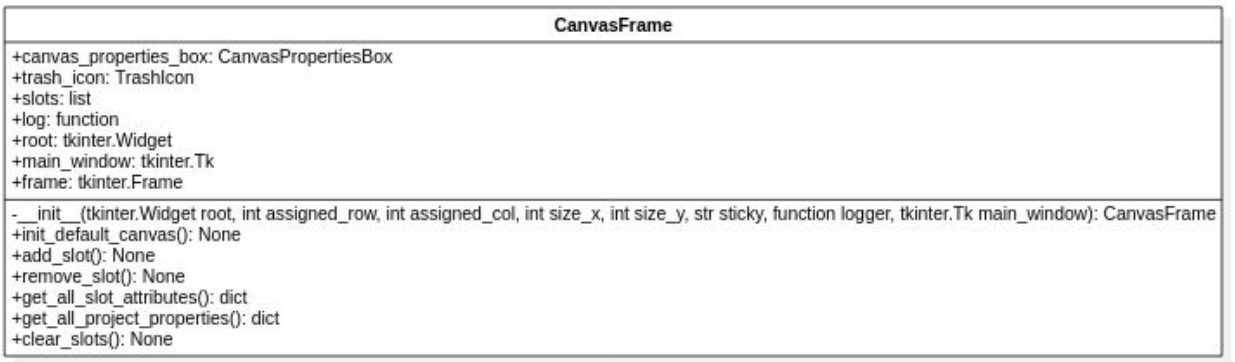

Figure 7 - CanvasFrame class diagram

Appendix E

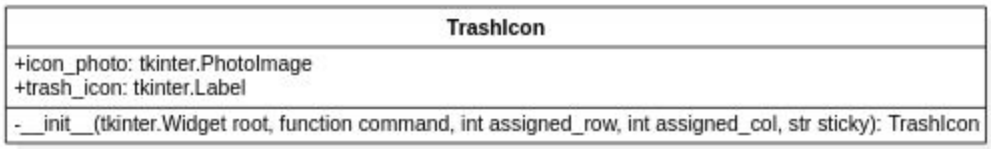

Figure 8 - TrashIcon class diagram

Appendix F

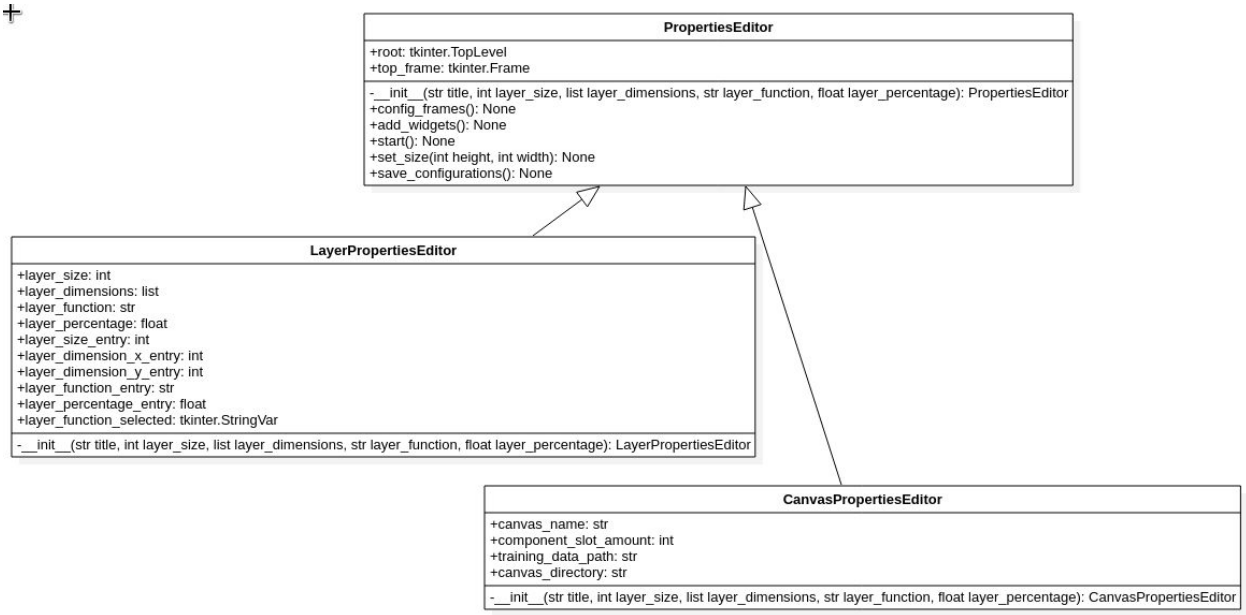

Figure 9 - PropertiesEditor class diagram

Appendix G

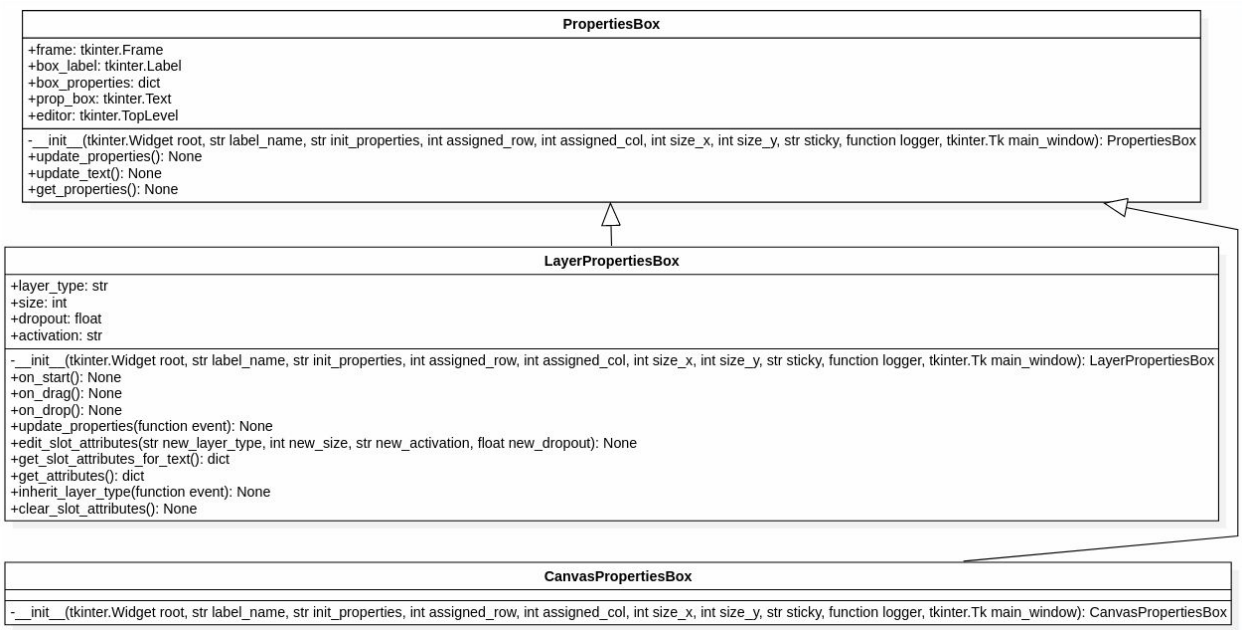

Figure 10 - PropertiesBox class diagram (Generalization link has been manipulated to promote legibility)

Appendix H

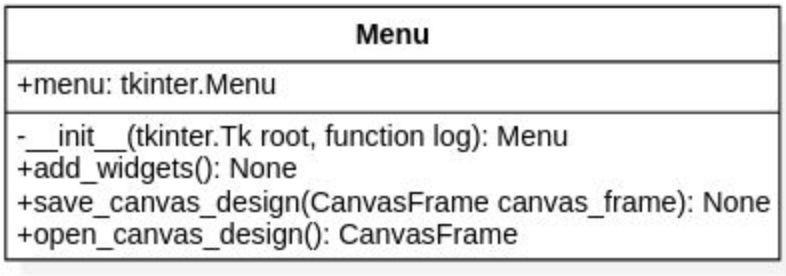

Figure 11 - Menu class diagram

Appendix I

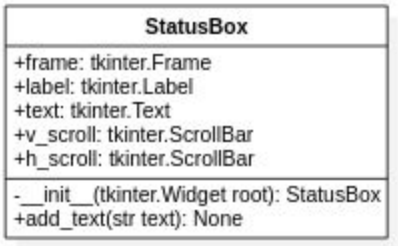

Figure 12 - StatusBox class diagram