CX1003: Introduction to Computational Thinking

Real-time Canteen Information System: NTU Smart Canteen

Group: CM1

Chua Zhi Loon James, U1921757B

Sean Dai Yun Shan, U1922832G

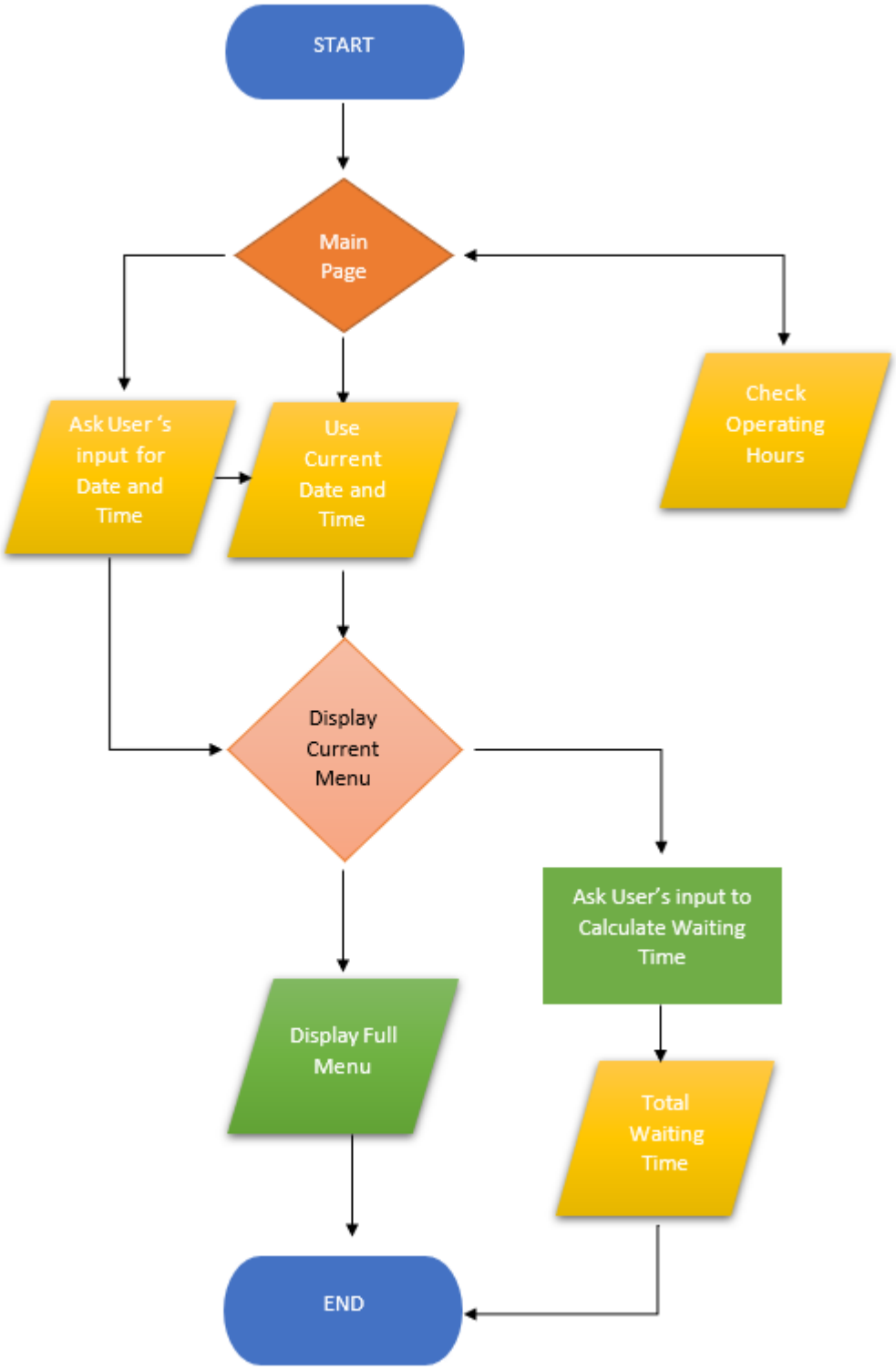
Elayne Tan Hui Shan, U1921730C

School of Computer Science and Engineering

Table of Contents

1 Algorithm Design

1.1 Top Level Flow Chart



1.2 Modules and User-Defined Functions

The application uses *os* and *datetime* modules to check existence of files and for the date and time functionalities it requires.

We created a function *OpenFile()* that checks the existence of the files before opening and returning the file object. In conjunction with “Menu Database” and “Operating Hours”, we are able to implement file handling.

The core functions are DataBase(), OperatingHours() and GetMenuByDayTime() which are used to convert data from the files into dictionaries for further processing and obtain the available menu based on user’s selected date and time. This allows the application to be flexible in accordance to the availability of food items that the stores only offer on certain days or times.

```
Store:  Vegetarian Food          2 mins      Price    Time          Day
Loh Mee                                      $3.00    All_Day       Mon
Laksa                                        $3.00    All_Day       Tue
Mee Rubus                                    $2.80    All_Day       Wed
Kway Chap                                    $3.00    All_Day       Thu
Prawn Noodle Soup                            $3.00    All_Day       Fri
Vegetarian Bee Hoon                          $2.80    Breakfast   Mon_Sat
```

Figure 1. Example of Food Items Provided only on Certain Days or Times

We have also created supporting functions that aid in developing the core functions such as CheckTimeInRange(). In addition, there are functions such as VerifyFile() and VerifyDayTime() that checks that the files are in the format that our application can use.

```
def ConvertTimeToSeconds(tTime : tuple) -> int:
    '''Convert tuple (Hour, Minutes, Seconds) to integer seconds for easy comparison'''

    return 3600 * tTime[0] + 60 * tTime[1] + tTime[2]


def CheckTimeInRange(tTime : tuple, tTimesRange : tuple) -> bool:
    '''Check given time is inside the start and end timings'''

    return ConvertTimeToSeconds(tTimesRange[0]) <= ConvertTimeToSeconds(tTime) <= ConvertTimeToSeconds(tTimesRange[1])
```

Figure 2. Code for Supporting Functions

These functions are collectively located in the “Functions.py” module to provide a clear separation between the algorithm and the Graphical User Interface (GUI) code.

2 Databases

2.1 Creation of Databases for Stores and Operating Hours

First, we collected information of the stores in North Spine Canteen. Then, we proceed to allocate the information into two files, the database for the stores under "*Menu Database.txt*" and the stores' operating hours under "*Operating Hours.txt*".

This approach is efficient as it allows us to modify the database easily. In addition, there are verification functions for the files so that we can reduce code errors due to incorrect storing of information in these files.

The menu database consists of the Stores and Food Name, Waiting Times, Prices, Available Times and Days. The other file consists of the operating hours and days for the stores. After we load in these databases, we convert them into dictionaries which can then be used and displayed on the GUI.

2.2 Reflections when Creating the Databases

We have learnt through CE1003 about functions to utilise file handling such as *open()* and functions to modify the data in these files using methods including *split()* and *readline()* to generate the dictionaries and lists. This allowed us to use various features such as strings and data operations.

On the other hand, the challenges we faced at the start include deciding on a method to efficiently store the information. Initially, we tried to hardcode the data but it was too demanding. For instance, this approach would be time-consuming if we had a large amount of information. Our current approach is to store the information into files which makes data editing easier. However, an improvement we could have done to make the data storage more efficient is to utilise Microsoft Excel and use CSV files and module.

3 Graphical User Interface (wxPython)

3.1 GUI Files

Our Panel uses the background image of NTU's Hive [1] for the application. It is displayed through *OnEraseBackground()* which was referenced online [2]. There are some changes to better suit our application.

```python
#Panel used in every page
class Panel(wx.Panel):
    def __init__(self, cParent : wx.Panel, sBgImgName : str) -> None:
        '''Constructor and loading background image'''

        super().__init__(parent = cParent, style = wx.FULL_REPAINT_ON_RESIZE)

        self.Bind(wx.EVT_ERASE_BACKGROUND, self.OnEraseBackground)
        self._sBgImgName = sBgImgName

        self._cBgImage = None

        #Only load background image if the file exists
        if os.path.exists("Bitmaps/" + self._sBgImgName):
            self._cBgImage = wx.Image("Bitmaps/" + self._sBgImgName)


    def OnEraseBackground(self, cEvent : wx.EraseEvent) -> None:
        '''Erase background and replace with image'''

        cDC = cEvent.GetDC()
        #Provide a default white background in case background image is not loaded
        cDC.Clear()

        #Only display background image if it is loaded properly
        if self._cBgImage:
            cClientSize = self.GetClientSize()

            #Perform scaling on image based on client window size
            cDC.DrawBitmap(self._cBgImage.Scale(cClientSize.GetWidth(), cClientSize.GetHeight()).ConvertToBitmap(), 0, 0)
```

Figure 3. Code for *Panel.py*

3.1.1 Main Handler

Running the main file, *MainHandler.py*, creates a *wx.App* that runs continuously and the application starts with *HomePage*.

```
#Only create application when handler is executed and not imported
if __name__ == '__main__':
    cApp = wx.App()

    #Load home page with background image name
    HomePage("HiveTrsp.png")

    #Start application loop
    cApp.MainLoop()
```

Figure 4. Code to Create *wx.App* and *HomePage Objects*

3.1.2 wxPython Widgets

wx.StaticBitmap() and wx.StaticText() are used to display information using images and labels. We used wx.Button() as the main input channel from the users and process them through button click events.

```
#Static textbox for displaying the food menu for the store
self._cStoreMenu = wx.StaticText(self._cPanel, style = wx.ALIGN_LEFT)

#Settings for the food menu text
self._cStoreMenu.SetLabel(sMenu)
self._cStoreMenu.SetFont(wx.Font(20, wx.TELETYPE, wx.NORMAL, wx.NORMAL))


''' Button Widgets '''


#Button for user to see full menu of the store
self._cFullMenuButton = wx.Button(self._cPanel)

#Button for user to get waiting time for the store based on number of people waiting
self._cCalcTimeButton = wx.Button(self._cPanel)

#Button for user to go back to the Home Page
self._cBackButton = wx.Button(self._cPanel)

#Button for user to exit the application
self._cExitButton = wx.Button(self._cPanel)
```

Figure 5. Code for wx.StaticText() and *wx.Button()*

We utilised *wx.BoxSizer()* to align and place all the widgets on the display. The widgets are added to the sizers via wx.Sizer.Add(). They are then displayed using the *wx.Panel.SetSizer()* and *wx.Frame.Show()* methods.

```
''' Horizontal Sizer to place first row button widgets '''


#Align the first row of button (widget) choices horizontally
self._cHoriSizer = wx.BoxSizer(wx.HORIZONTAL)

self._cHoriSizer.Add(self._cSetDTButton, 0, wx.RIGHT | wx.CENTER, 20)
self._cHoriSizer.Add(self._cUseCurrentDTButton, 0, wx.CENTER)


''' Vertical Sizer to place widgets '''


#Align the widgets vertically
self._cVertSizer = wx.BoxSizer(wx.VERTICAL)

#Add space from the top to the widgets
self._cVertSizer.AddStretchSpacer()

self._cVertSizer.Add(self._cHeadingText, 0, wx.ALL | wx.CENTER, 20)
self._cVertSizer.Add(self._cDTText, 0, wx.ALL | wx.CENTER, 10)
self._cVertSizer.Add(self._cHoriSizer, 0, wx.ALL | wx.CENTER, 10)
self._cVertSizer.Add(self._cCheckOHButton, 0, wx.ALL | wx.CENTER, 10)
self._cVertSizer.Add(self._cExitButton, 0, wx.ALL | wx.CENTER, 50)

#Add space to the bottom of the widgets
self._cVertSizer.AddStretchSpacer()


self._cPanel.SetSizer(self._cVertSizer)
self.Show()
```

Figure 6. Code for *wx.BoxSizer(), wx.Panel.SetSizer() and wx.Frame.Show()*

Every frame of the application has a “Back” button to allow users to return to “previous pages”. These buttons clicks are linked to functions such as PressBack(), an event handler function (EHF) that calls wx.Frame.Hide() to close the current frame. Next, a new frame is created for the previous page content.

```
def PressBack(self, cButtonEvent : wx.CommandEvent) -> None:
    '''Go back to the Start Page when user clicks on Back button'''

    #Placed here instead of the top to avoid infinite importing error
    from StartPage import StartPage

    self.Hide()
    StartPage(self._sBgImgName, self._sSelectedDT)


def PressExit(self, cButtonEvent : wx.CommandEvent) -> None:
    '''Close the application when user clicks on Exit button'''

    self.Destroy()
```

Figure 7. Code for Button Events

In addition, *PressExit()* stops the application when users click on the "Exit" button found in all the frames. Hence, users can quit the application when they wish to.

3.1.3 Home Page

HomePage displays the system's name and the current date and time. Its buttons also allow users to set a specific date and time, list available stores and display stores' operating hours respectively.

3.1.4 Date and Time Setting Page

SetDTPage utilises *wx.adv.CalendarCtrl()* and *wx.adv.TimePickerCtrl()* widget to allow users to select a specific date and time.

The "Confirm" button links to *StartPage* with the selected time and date.

The "Use Current Date and Time" button directs to *StartPage* with the current time and date instead of the selection.

3.1.5 Available Stores Pages

StartPage obtains the available menu according to the selected date and time through *GetMenuByDayTime()*. Buttons are generated to view each of the store menus on *StorePage*. If no stores are available, the operating hours will be displayed instead. *StorePage* displays the available menu based on the selected store. The “See Full Menu” button directs to *FullMenu* and displays the full menu via *GetFullMenu()*. The “Calculate Waiting Time” button directs to *CalcTime* and provides an estimated waiting time.

3.1.6 Estimate Waiting Time

CalcTime provides a text field, *wx.TextCtrl()*, to prompt users for input on the number of people queuing. When users click the “*Confirm*” button, the input is validated, then sent to *CalculateWaitingTime()*. There will then be a display of the waiting time or an error message for invalid user inputs.

3.2 Error Handling Test Cases

wxPython’s calendar was used to reduce input errors as users can only select the date and time from the options. It also has a default value if users select nothing. Hence, we can use this input without the need for validation.

For the estimation of waiting times, we utilised if-statements to check that the input is numeric using *str.isdigit()* and that it is less than 100 people. This large amount of people is unrealistic. In addition, the condition eliminates the processing of very large user inputs that may crash the system. Finally, error messages are also displayed to inform users when their inputs are problematic.

3.3 Reflections for GUI

Originally, we implemented the system under a single frame but it was hard to comprehend. Knowledge obtained from CE1003 reminded us of the principles of using functions. Subsequently, we applied this to our implementation and disperse the frames into different classes.

CE1003 has taught us on the importance for validation of inputs and we have implemented them for the estimation of waiting times.

```
sNumber = self._cTextField.GetValue()

if not sNumber:
    self._cTimeText.SetLabel("Nothing was keyed in.")

elif not sNumber.isdigit():
    self._cTimeText.SetLabel("Invalid number of people.")

elif int(sNumber) > 99:
    self._cTimeText.SetLabel("Too many people in queue.")

else:
    self._cTimeText.SetLabel("Waiting Time: " + CalculateWaitingTime(self._dDataBase, int(sNumber), self._sStoreName))
```

Figure 8. Code for Validating User's Text Input

Insights gained from CE1003 also guided us in using appropriate data types and structures for various circumstances. Methods like indexing and slicing came into use and made it easier to process and generate data.

```
#Verify food item line is in correct format
if len(lData) == 4 and VerifyPrice(lData[1]) and VerifyDayTime(lData[2]) and VerifyDayTime(lData[3]):
    dDataBase[tStoreKey][lData[0]] = (float(lData[1][1:]), lData[2], lData[3])
else:
    print("Invalid Line in File: " + sLine)
```

Figure 9. Code Showcasing Strings, List Indexing and Slicing

4 Individual's Contributions

Sean Dai	❖ Program Design and Flow Chart ❖ Created both store and operation hours databases ❖ Converted databases into useable functions
James Chua	❖ Created supporting functions and processed databases to provide outputs for GUI ❖ Added bitmaps for all buttons
Elayne Tan	❖ GUI Modules - Code and Design

References

[1] S. Lim, "NTU emerges in top 50 of Best Global Universities Rankings for the first time," 1 November 2018. [Online]. Available: https://static.businessinsider.sg/2018/10/NTU-the-hive.jpg.

[2] Mike, "wxPython: Putting a Background Image on a Panel," 18 March 2010. [Online]. Available: http://www.blog.pythonlibrary.org/2010/03/18/wxpython-putting-a-background-image-on-a-panel.