NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ1003 Report Assessment

Neo Guat Kwan

Ng Chi Hui

Nguyen Linh Lan

School of Computer Science and Engineering

Table of content

1. Algorithm Design

PyQt5 was chosen for building the GUI app because its complementary application, QT designer, has a simple drag-and-drop interface. Furthermore, the designer could generate a UI file, allowing us to separate any changes in design from our functions. Our program flow is outlined below (Figure 1).

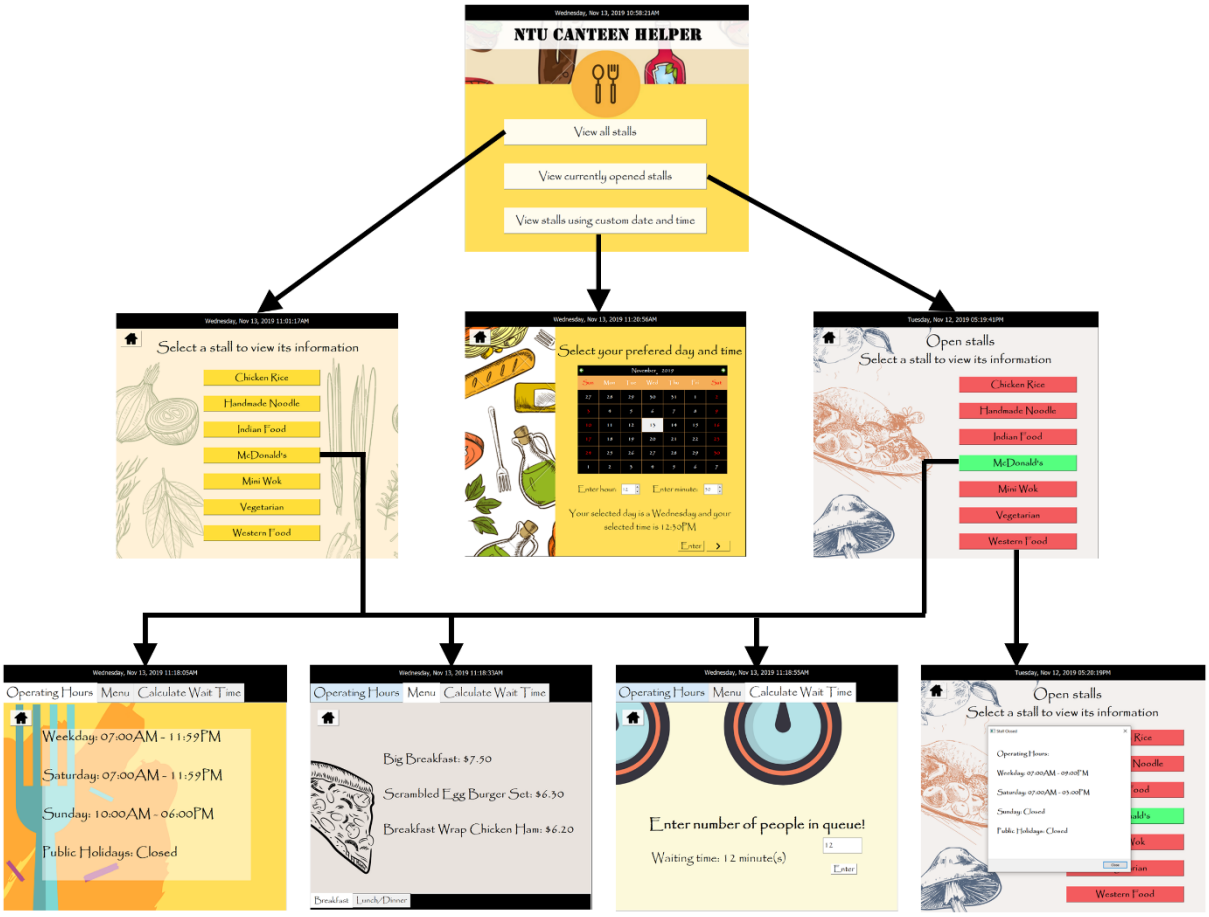


Figure 1: A flowchart showing the working of our application

2. User Defined Functions

a. find_open_stalls (day, date_time)

This function identifies the open stalls based on the specified day and datetime. Using a for loop to iterate through all the stalls in our operating hours database, the operating time range of the specified day will be retrieved. If this retrieved content is the string 'Closed', it moves on to the next stall. However, if the content is a time range, it checks whether the datetime argument is within this range. If it is, it appends the stall to the open_stalls list but if not, the loop continues. After iterating through all the stalls, the open_stalls list is returned.

b. get_now_menu (stall, day, date_time)

This function retrieves the menu of a stall at the specified day and time. The function first identifies which type of menu the stall has. If the stall has no special menu, it returns the only menu available. If the stall uses a daily menu, it returns the menu of the day specified. If the stall uses a 'time' menu (breakfast/lunch) it will then check whether the time specified is within the breakfast time range. If it is, the breakfast menu is returned and if not, the lunch menu is returned.

c. info_reg (self, stall),
info_day (self, stall),
info_time (self, stall),
info_current (self, stall, day, date_time)

As a slightly different layout is required for presenting the different types of menus (Figure 2, Figure 3, Figure 4), separate information pages were created for each menu

type. The info_reg, info_day, info_time and info_current functions bring users to the respective information pages and populate the empty labels with information of the specified stall. Similar to the info_reg function, the info_current function brings users to the same page but instead of displaying a regular menu, it calls the get_now_menu function and displays the current menu, regardless of menu type (Figure 5).

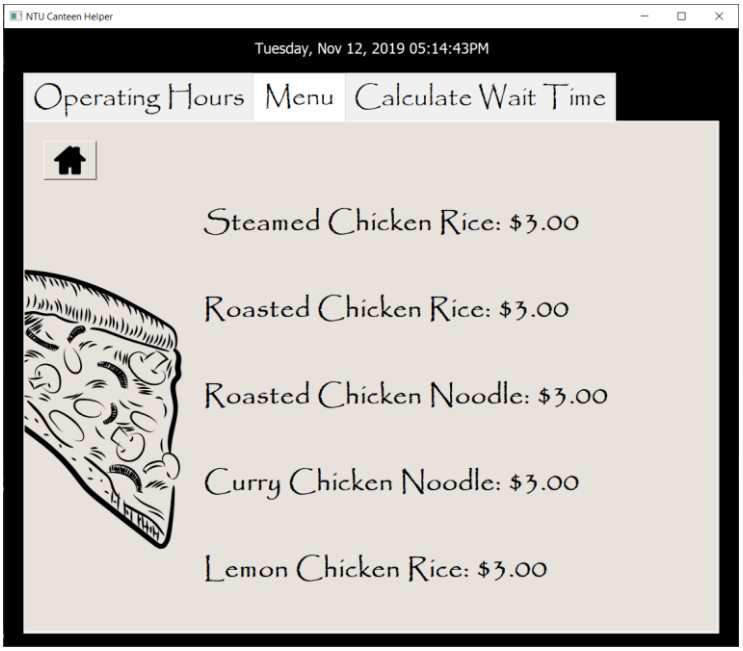


Figure 2: Layout for stalls with a regular menu

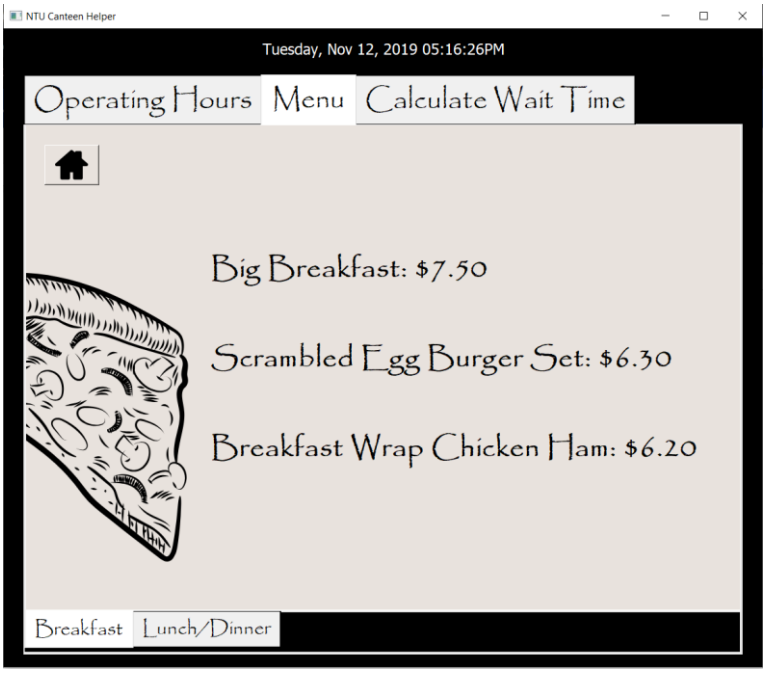


Figure 3: Layout for stalls with a periodic menu

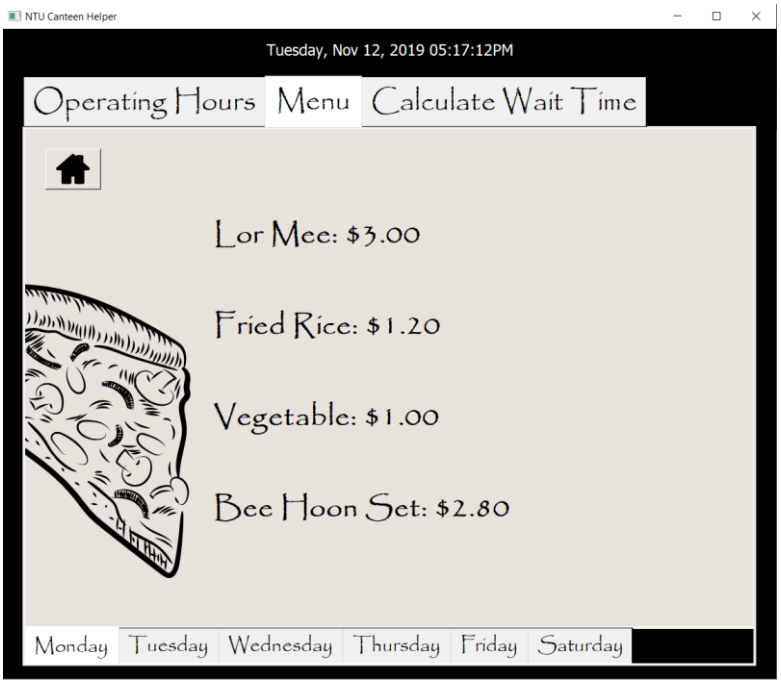


Figure 4: Layout for stalls with a daily menu

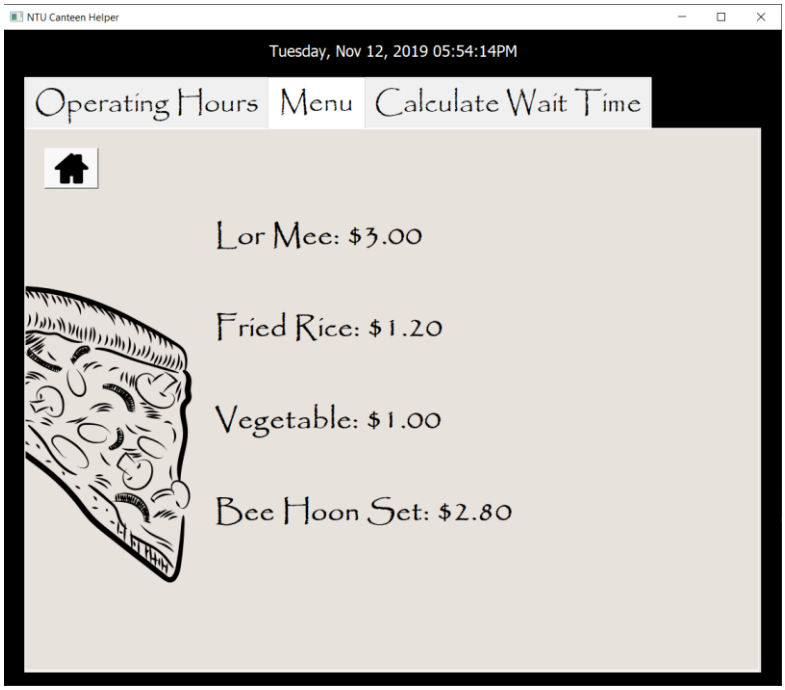


Figure 5: Display of the current (Monday) menu of vegetarian stall by info_current function

d. open_stall_btns (self)

This function brings users to the stall selection page and sets the colours and functions of the stall buttons based on whether it is open or closed at the given day and time (Figure 6). After retrieving the list of open stalls from list_open_stalls function, a for loop checks whether each stall is in the list to determine their opening status. Open stalls have their buttons set to green and connected to the info_current function while closed stalls have their buttons set to red and connected to the close_msg function, which prompts a message box (Figure 7).

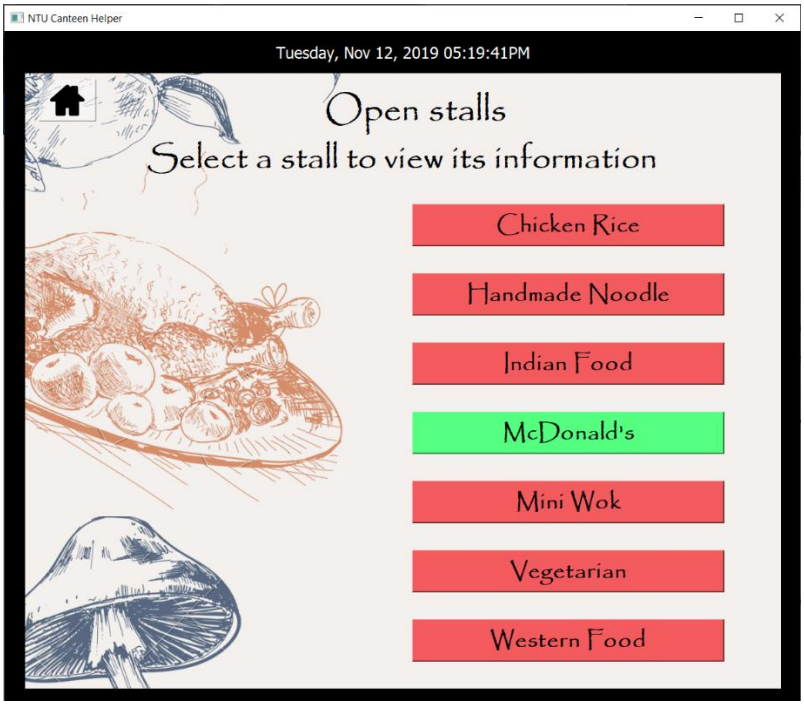


Figure 6: Buttons set to reflect stalls' open/closed status

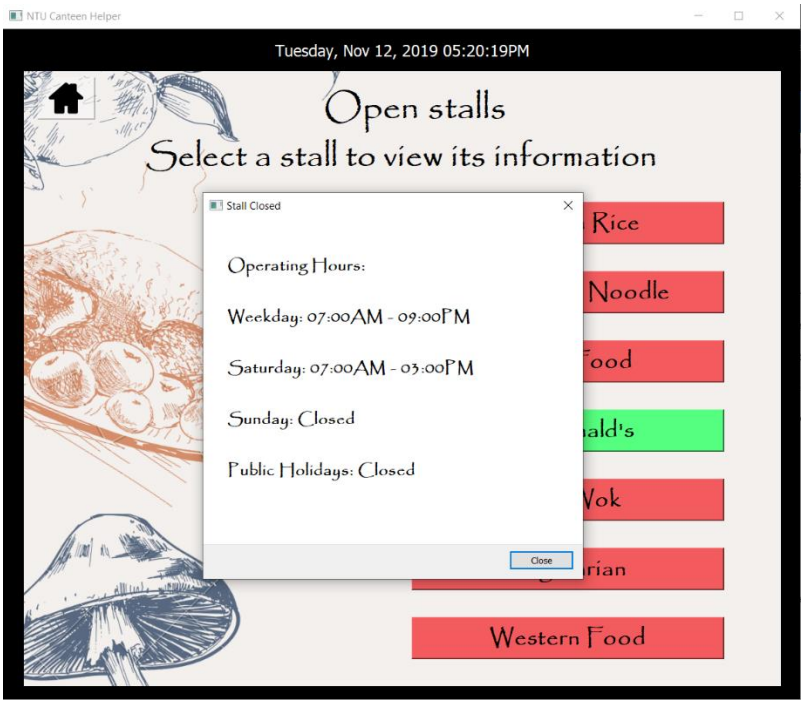


Figure 7: Message box appears when a closed stall is selected

e. wait_time_check (input, error)

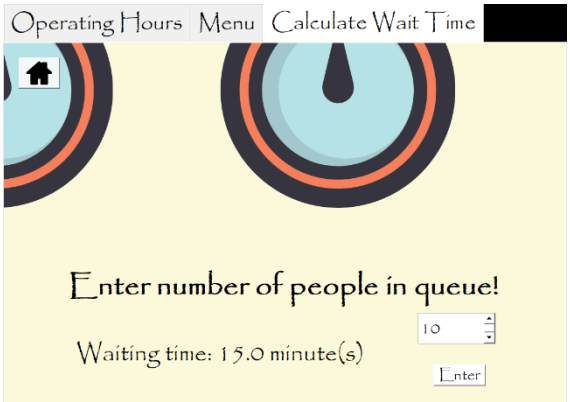
This function uses try and except to check the user's input for number of people in queue. It tries to convert the user input into an integer and multiply it by the wait time of the respective stall. If there are no errors, the label is set to display the calculated waiting time. Otherwise, an error message will be displayed.

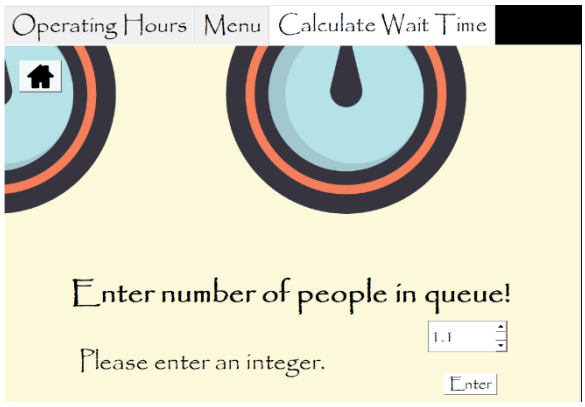
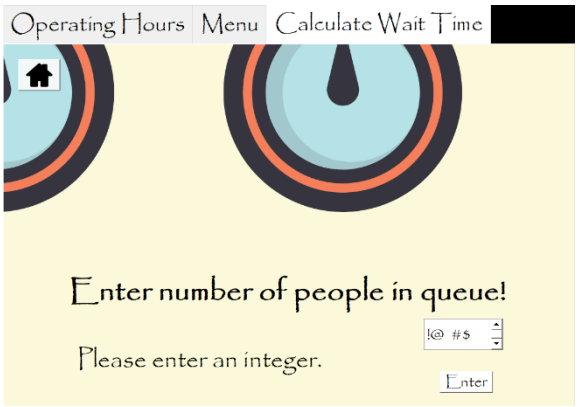
f. customised_datetime (self)

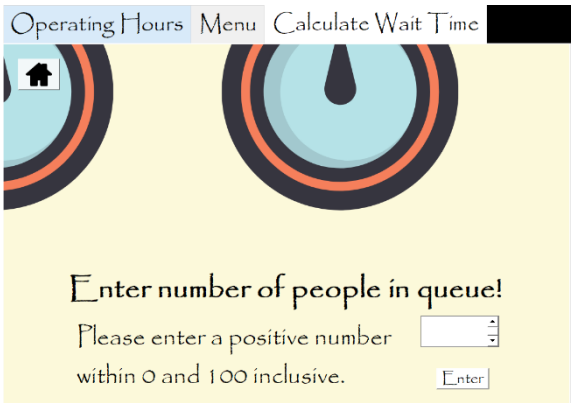
This function gets the user's custom date from the calendar widget and time from the spin boxes. The date and time are combined to form a datetime object and the alphabetical day is derived from the datetime. The datetime and day are then assigned to the global variables datetime_now and day_now respectively so that stall information will be displayed based on these custom values.

3. Error Testing

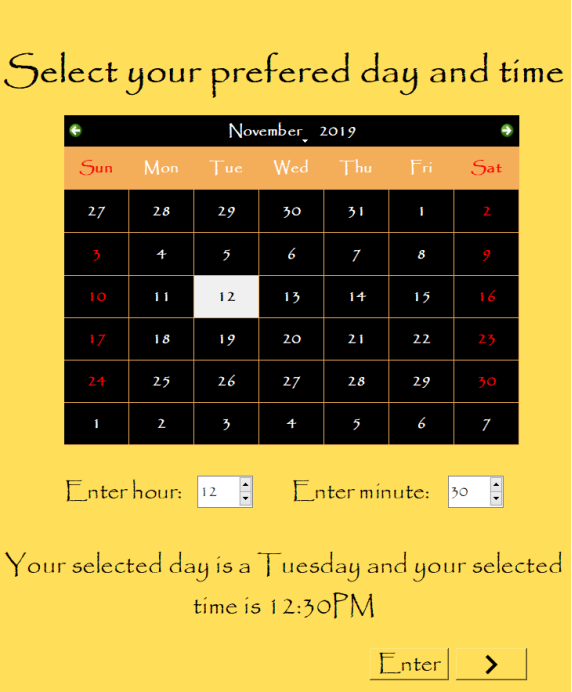
a. Wait Time User Input

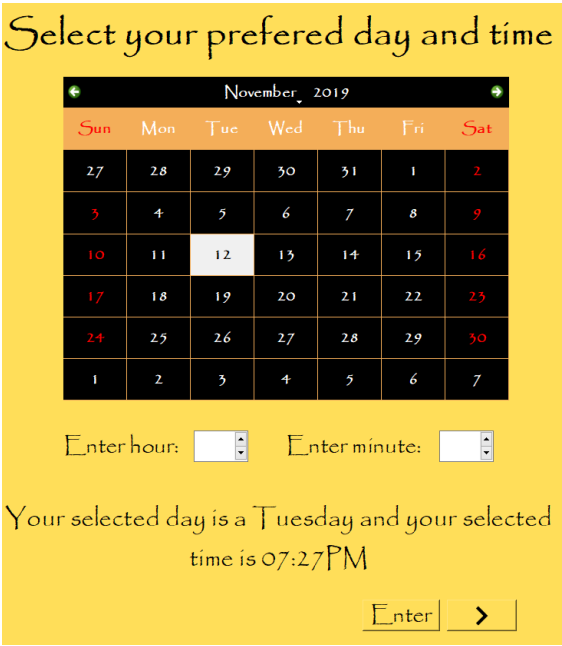
Input Type	Test Cases	Examples
Good	**Input:** ✓ An integer ✓ Value between 0 and 100 **Output:** The calculated waiting time.	 **Figure 8: Entering a good input**
Bad	**Wrong Data Type** **Input:** × Alphabets × Float value × Special characters **Output:**	 **Figure 9: Entering alphabets**

	Error message (Figure 9, Figure 10 and Figure 11)	 **Figure 10: Entering float value**  **Figure 11: Entering special characters**

Bad	**Out of Range** **Input:** × Value greater than 100 × Negative value **Output:** Error message (Figure 12).	 **Figure 12: Entering out-of-range values**

b. Custom Date and Time Input

Input Type	Test Cases	Examples
Good	**Input:** ✓ Hour value between 0 and 23 ✓ Minute value between 0 and 59 ✓ Year between 2019 and 2029 **Output:** Displays the selected day and time (Figure 13).	 **Figure 13: Entering a good input**

Bad	**Wrong Data Type** **Input:** × Alphabets × Float value × Special characters **Output:** Any characters other than numbers cannot be typed. If any field is left blank, the default value is the current date and time (Figure 14).	 **Figure 14: Current date and time are used when fields are left blank**

Bad	**Out of Range** **Input:** × Hour value greater than 23 × Minute value greater than 59 × Year outside range of 2019 to 2029 **Output:** For hour and minute inputs, any digit that makes the total value go beyond the limit cannot be typed. For the calendar input, a value outside the range is replaced by the last selected value (Figure 16).	 **Figure 15: Any digit that causes value to exceed limit cannot be typed.**

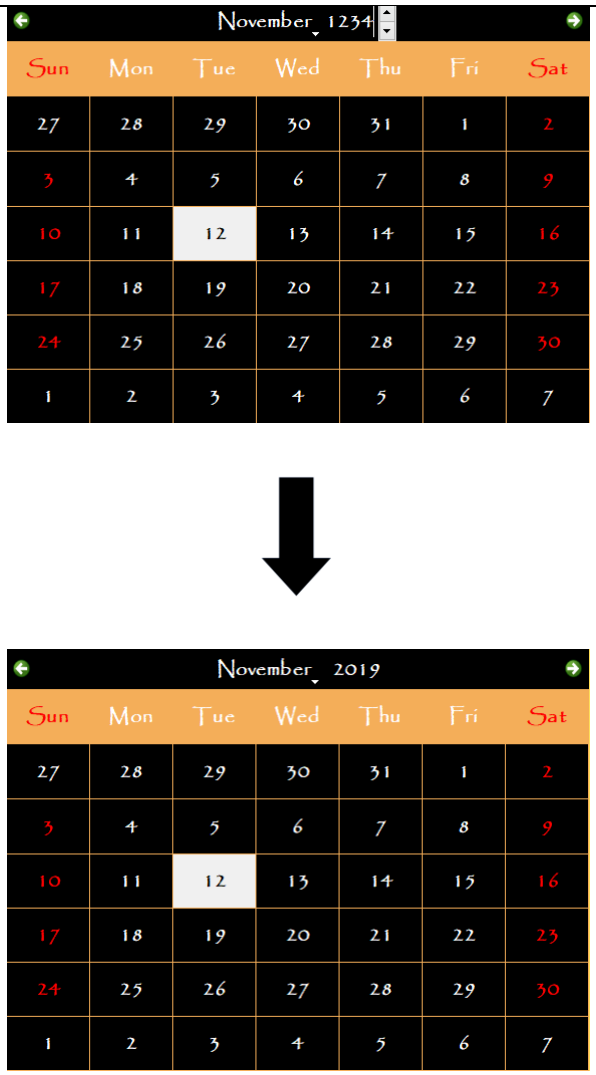


Figure 16: Entering a year outside valid range

4. Reflection

Initially, we wanted to develop a GUI directly. However, unsure of where to begin, we decided to develop the console version first since the interface was simpler. After completing the console, we realised that the algorithms could be applied to the GUI with some modifications to the handling of inputs and outputs. For example, the console version used number inputs for selection and printed out the outputs while the GUI used buttons for selection and display outputs on labels. Developing the console also helped us familiarise with the program flow, expediting the GUI development. Hence, we learnt that given a difficult task, it might be better to tackle the smaller problems first before addressing the bigger problem.

While debugging, we also faced challenges in searching for solutions online due to our limited understanding of object-oriented constructs. Online solutions often contained unfamiliar concepts such as classes, which impedes code comprehension. Furthermore, due to the specificity of the answers, we had to extract the parts pertinent to our problems. To overcome this, we often used trial-and-error approach, whereby modifications were made until the problem was fixed.

Besides, at the beginning of our project, we mainly focused on achieving the intended outcome. Consequently, functions sometimes contain similar blocks of code. To increase the modularity of our functions, we made use of parameters. For instance, in our find_open_stalls function, we utilised the parameters ‘day’ and ‘datetime’ so that only one function was needed to find open stalls for both current and custom date and

time. When encountering code duplication, we took out the identical parts and made it into a separate function. These practices helped shorten our code and improve its readability and cleanliness significantly.

For future development, we could use csv file to store our data to ease the process of data modification, especially for a large dataset. Additionally, we would like to include functions that can dynamically generate buttons based on the number of stalls in the database. Besides, since our app is currently native, we could integrate the use of cloud server to source public data.

5. Contribution

Name	Guat Kwan	Lan	Chi Hui
General contribution	Overall algorithm and flow	GUI	GUI, database
Function contribution	• Display information and menus based on current date and time	• Event handling • Display information and menus based on custom date and time	• Data retrieval • Waiting time calculation and exception handling

Table 1: Contribution table of our group member