CSC3002 Project Report

1. Introduction

The project implemented a software application that simulates some basic functions of a real OS by using C++ languages and Qt widget tools. It simulates the memory allocation, file system, task scheduling, and error system and includes the functions of booting up, login in, users' accounts, calendar, calculator, text editor, memo, media player, video game, browser, camera, painter and photo. Moreover, it enables users to check the real-time condition of CPU and storage in the monitor.

Compared with the proposal, some changes are made:

1. Originally, the proposal only tries to read a specific folder and do some changes in this folder. However, our project now can read any folder chosen by the user and edit any text file. It can make the fake OS more useful.
2. In the proposal, we designed a very simple task scheduling that no more than two tasks can be run at the same time. Now, our project uses the Fist In First Serve rule. When the memory allocation is nearly full, the project will display a warning and force the user to close the earliest task that he or she opens.
3. We proposed to borrow the memory allocation system from a real OS and display the information in the proposal. However, we find it hard to get the concrete memory usage for the task in our project. We can only gain the whole memory usage of the project, but we cannot distinguish how much memory the video game or the media player in our project used. Therefore, we finally decided to fake a memory allocation system. The memory allocated for each task refers to the real OS and the memory usage is dynamic, so that the fake memory allocation is more realistic.
4. In the project, we mentioned a general idea about users' accounts in our fake OS. In the present project, we fulfilled our ideas and add startup video, boot sound, and login process to simulate the starting up in real OS. Besides, users can add or delete accounts after they successfully log in. It enriches the account task.
5. In the proposal, text editor, calendar, calculator and video game have been fulfilled. Besides, the project adds some new functions, including memo, media player, browser, camera, painter, and photo. Details of these tasks will be explained later.

2. Related Work

1. QT Class used for Media Player, Camera, and other tasks: https://doc.qt.io/qt-5/index.html
2. Sample Operating System for reference: https://demo.os-js.org/
3. QT Creator Quick Start. YaFei Huo. Beihang Univeisity Press. Third Edition.
4. Double Buffering | C++ GUI Programming with Qt4：https://www.informit.com/articles/article.aspx?p=1405227&seqNum=4
5. Creating a Qt Widget Based Application：https://doc.qt.io/qtcreator/creator-writing-program.html
6. Basics about GUI layouts, how to implement a customized button with picture:

https://www.bilibili.com/video/BV1Nb411K7bW?p=44

7. Overriding mouse events, pumping out widgets when mouse moved to certain places: https://www.cnblogs.com/liuruoqian/p/12044571.html
8. Use of QPainter http://shouce.jb51.net/qt-beginning/21.html
9. Use of QTimer https://wizardforcel.gitbooks.io/qt-beginning/13.html

3. Our Work

1) Memory allocation:

In the fake OS, we initialized the memory for each application and task and the whole OS has 4GB memory. The fake memory allocation includes many details. First, to make the memory allocation more realistic, the initial value of memory refers to the value in the real OS. For example, the memory for the calculator is just 300MB, while the video game is 1GB. Second, each operation done by users in any task would be considered. For example, when the window of the video game is just open, the memory usage is much smaller than the situation that users begin to play video games. Third, the memory allocation for open tasks is dramatic and it would have minor fluctuations every second. Lastly, the fake OS prevents the memory usage from being more than 4GB. All the information about current memory allocation is displayed in the monitor task that would be explained later.

2) File system:

The file system can read and process file folders and files. By clicking the “finder” button, the user can choose which file folder they want to open. At the same time, they can rename, create, delete the file folder as well. After opening the file folder, all the files under this folder (including those in the subfolders) can be read, and they are then listed in one table displayed on the screen. The information including file path (absolute), file name, file size, and create time are also shown clearly. Users can rename or delete the file by double click the file in the table. The system will ask the user again to continue the process because this change is permanent. After confirming the operation, the system will also tell the user whether the process is successful or not. All the operations done in the system would change the original file.

3）Task scheduling：

For a real OS, it has an algorithm to schedule running tasks to optimize CPU and avoid potential errors. Our fake OS simplifies the task scheduling. Firstly, there are two display boards on the main window, one of them shows the tasks that are running, another outputs the information when a task is open or closed. All of the tasks are set with two states, running or not running. Secondly, the tasks opened are put into a queue one by one. When the task is terminated, it is erased from the queue, and its state is also changed. By monitoring the internal memory that has been used, the first opened task will be terminated automatically if the internal memory is almost full. This can

help prevent running out of memory to some extent. Thirdly, a forceful closing function is implemented, allowing the user to terminate the specific task in the monitor.

4) Exception system:

The exception handler in a real operating system can be invoked in many cases. In the fake OS, we consider several cases.

- Memory Error: When Out of Memory Error occurs, the fake OS would display a warning to remind the user. Besides, it would force the user to quit tasks till the memory usage is under 4 GB.
- Save Error: When the user tries to save files into an invalid path, the fake OS would ignore the operation and keep it operating normally. In some specific application, such as photo and painter, the system will prompt n warning message if the file saving fails. If the path address is empty, the fake OS will display a warning to remind the user to enter the path.
- Division Error: When the fake OS tries to do zero division，the fake OS will display a warning to remind the user.
- Bad Access Error: When the user tries to close the Media Player window with the playing media, the bad access error will occur. To avoid this kind of crash, the player will be deleted when the window is closed.

5) Interfaces：

Our project includes three major interfaces:

Booting up:

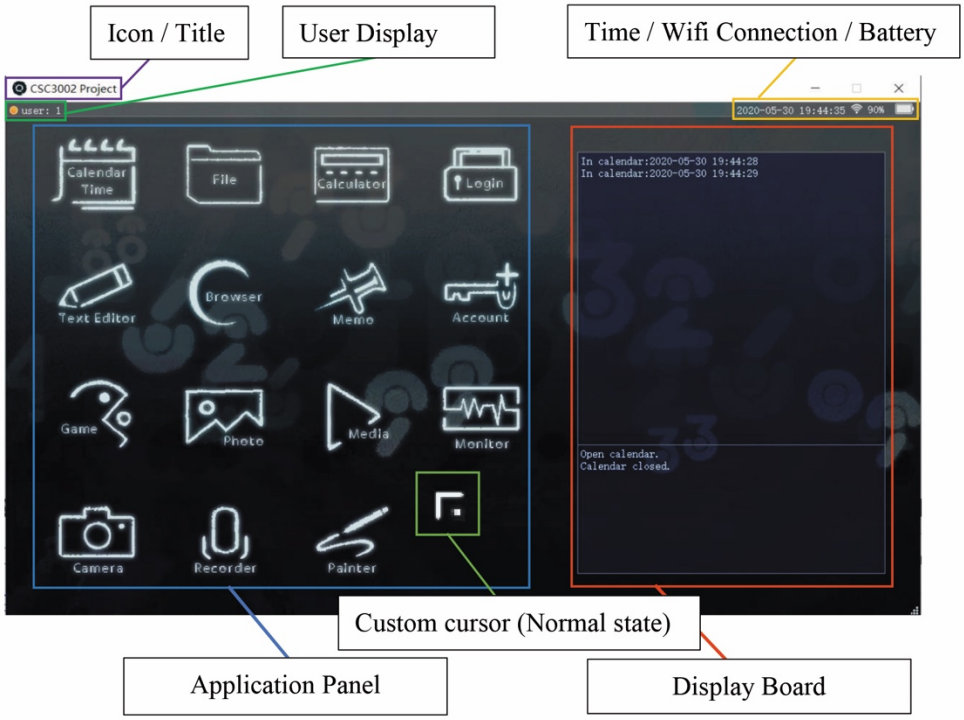
The booting interface, which automatically appears at the very begging when the program is executed. When this interface appears, users cannot use the cursor and must wait until booting up is finished. The animation displayed are not video or gifs, but animation units which computes its

motions on spot during execution. After the animation finishes the login interface would automatically appear. This interface simulates the powering up of an OS system.

Login:

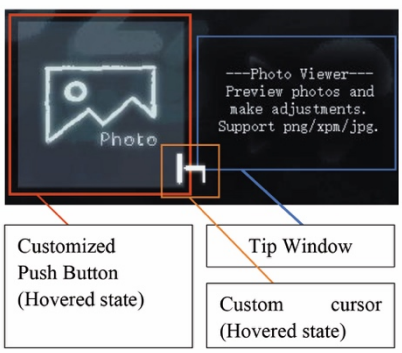
The Login part refers to the real OS. The Login task shows up after a boot animation. Users need to input a name and its related password to enter the fake OS. Only the right account will be accepted. Otherwise, the fake OS would give the user a warning and ask him or her to enter information again. To implement this part, all the information about accounts is saved into two text files, password file and username file, with one-to-one correspondence. The fake OS has at most ten accounts. When the user enters a username and password, the fake OS will check the information against a password file and username file. If the user enters the correct account, the login will accept the information and the main window will show up.

Main Interface:



The main interface can be mainly divided into 2 sections, the action panel and display board. The action panel contains 15 customized pushbuttons, each of them can call a unique application; The pushbuttons are displayed by ratio to the main window, not exact locations in pixel. This allows it to be able to rescale if the main window is dragged & reshaped. The display board shows the user's interaction with this operating system. It would display in time which application is called, is being operated, and is killed. On the upper side of the main interface is a bar that provides some general information: The username (which would be the same name entered in the login interface), current time display, WIFI connection state, and battery usage. In Windows system, the WIFI logo will be changed according to the real WIFI state on user's own computer. In MacOS system, the WIFI logo is set to be connected all the time. In both Windows and MacOS system, the battery would decrease with time. The cursor image would be changed to a unique one which suits to the UI design of this Operating System.

Customized Pushbuttons (Used in main interface and booting interface)



Each push button located in the action panel is an object of MyPushButton class, which inherits from QPushButton and extends its features:

- The pushbutton uses an image as its outlook. (Pixmap)

The pushbutton changes its outlook when cursor hovers over or clicks it. (StyleSheet)

- The pushbutton performs animations and sound effects when being clicked. (QAnimation, QSound)
- The cursor changes its outlook when hovering over these push buttons. (QCursor)
- The pushbutton displays a small widget aside of it when the cursor enters its region and hides it when the cursor leaves. This widget shows tips explaining the function of that exact application. (QEvent, QWdiget, QLabel, QString)

The animations shown during booting up are also made up of MyPushButton class objects. By setting the 'AsButton' parameter to false, these objects have their tips window, color change, and cursor change function disabled, only able to perform bouncing and shinning animations. They are used as animation units instead of pushbuttons.

6) Account：

The Account task is used for creating and deleting accounts. When the user successfully login, the user can create a new account. If the account has existed or there are already ten accounts, the account creation will fail. For deleting accounts, the user needs to enter a secrete password. If the secrete password is wrong, a warning will be given. The user can delete several accounts by only one operation and click the refresh button to see the updated accounts. To implement this part, all the information about accounts is saved into the password file and username file. When a user creates or deletes accounts, the content of these text files will be changed relatively. The refresh button is to update account information by scanning through the two files.

7) Calendar：

The calendar app can show the current time and a calendar, it also includes a timer function. The current time is updated per second. To implement the task, we get the current time and show it using a Qtimer with a 1s interval. For the calendar, the basic calendar widget in Qt is used. Moreover, Qtimer also used to implement the timer. The Qtimer starts when the start button is clicked, it is stopped when the stop button is clicked. Therefore, after clicking the stop button, the eclipsed time can be recorded and shown to the user.

8) Calculator：

The calculator can perform addition, subtraction, multiplication, and division operations. Users can click the buttons on the screen or press the keyboard to enter the numbers and

symbols. After clicking the '=' button, the result will be shown. if the user inputs some symbols that cannot be dealt with, it will show an error. To implement this task, we accept the user's inputs in an infix expression, then turns it to postfix expressions. After that, a stack structure is used to calculate the result.

9) Text Editor:

The text editor is a tool that can edit a text file. Users can choose any text file from their computers and edit its content. Besides, the text editor has basic functions, such as redo, undo and save. If the user quit the text editor and there is some change in the content, the program will remind the user of saving the content. To implement this part, we read the whole content and display it based on the path that the user chooses. If the user clicks the save button, we read the new content in the Text Edit and replace the original content in the text file. Besides, the undo and redo operation is based on the corresponding function of Text Edit. When the user directly closes the Text Editor, we compare the content in Text Edit and original content from the text file. If there is any change, we remind the user to save the file.

10) Browser:

By inputting URL, the user can go and visit the website, to implement this task, we make use of the browser on the user's own computer to get access to the target website.

11) Memo:

The memo allows the user to write a note or short information with time, to remind himself what he is going to do. User can read all the recorded memos, the matters that has happened will not be shown. It also provides a function that users can search the memo on a specific date. To implement this task, all the memos are saved in one text file named "matter.txt", when the user sets the memo, the information will be written into the text file. If the user wants to search the memo, the information can be read from the text file and displayed.

12) Media Player

The Media Player can play both music and video. The Media Player has its playlist and users can add new media or remove current media. There is a slider related to the current media, users can move the slider to change the pace of current media. Users can also choose to play, stop, or pause the media. Besides, the basic information of current media is on the title of this window, such as media name and author. To implement this part, this Camera application relies on the Class QMediaPlayer. Simple operations like play, stop, and, pause is from the class. When users remove current media, the program will remove it from the playlist. When users add current media, the program will ask the user to choose the path of the new media and add it to the playlist. Windows can play WMV MP3 formed file, MacOS can play MP4 MP3 formed file.

13) Monitor:

The monitor shows the real-time state of every task. It includes two-part, CPU, and internal storage. All tasks are listed in two tables in order. For example, if the video game is opened and used most internal memory, it is listed in the first row of the table. The tables are updated per 3 seconds. What's more, in the internal storage part, the total used memory is shown in a percentage form. Besides, the user can terminate tasks in the CPU part by double-clicking the task, this operation is only allowed when the task is opened. To implement those functions, several global variables are used. For example, when a task is opened, its flag becomes 1, a pointer points to the task instance is stored. The monitor can get the state and change the information, it can also terminate the task using the pointer.

14) Camera

The Camera can help users take pictures and videos. Before taking pictures and videos, users must choose the path for later pictures or videos he or she will create. Otherwise, a warning would be given. After that, users can click the shot button to take pictures or click the start and stop button to take videos. The duration of the video is on the title of this window and the picture or video will be saved automatically into the path when the stop or shot button is clicked. To implement this part, this Camera application relies on the Class QMediaRecorder and Class QCamera. Shot and Take videos are from these two classes.

15) Recorder

The Recorder can help user record sound. Before recording, users must choose the path for the later sound file he or she will create. Otherwise, a warning would be given. After that, users can click the start and stop button to record sound. The duration of the sound file is on the title of this window and the file will be saved automatically into the path when the stop button is clicked. To implement this part, this Camera application relies on the Class QAudioRecorder. Recording operation is from this class.

16) Painter:

The painter is a drawing board, the user can draw pictures using a mouse in a specific area. It enables the user to change the width of the pen and freely modulate the color with the palette. After finishing the drawing, the user can save the picture in png form and name the picture in the "Save Image" dialog box. It will give the user a success message if it is saved, and fail message otherwise. To implement this task, we track the press, move, and release the event of the mouse, drawing lines using the last point and endpoint.

17) Gluttonous Snake:

The Gluttonous Snake is a leisure puzzle game where players can use WASD to control the movement of the snake. After entering the game, players need to click the "Start" button in the message box to start a new game. During the playing, it enables players to check their score in real-time and press P to pause the game. The instructions and score will be displayed on the right area of the game interface. When the game is over, a settlement message will inform your score and give you the option to start a new game or quit the game. To implement the game, QWidget class was inherited and the paintEvent and keyPressEvent functions were overridden to build the connection between users' input and the change of game screen. Functions that implement the snake actions, random food production, and snake-alive detection were added to the game class. Moreover, a QTimer object was created to update the game interface at certain intervals to implement the snake actions.

18) Photo

We designed this application to allow users to load, view, adjust, and save images in this operating system. In the aim to make it easier for users to view their photos, all operations are put into the context menu such that users can enjoy their picture to the full screen.

Operations includes loading image, zooming image, brightness, and saturation adjustment, back to original and save. This application can read the png, XSL, and JPG format images stored on the computer. When loaded in, this application would measure the size of this image and display it to the maximum size possible to place the entire image inside the widget. The wheel event and mouse press event has been overridden such that users can also use the wheel to zoom in and out and press & hold to drag images across the display. The moving and scaling changes some private variables of the Photo class and updates the piantEvent, which taking in those new arguments would paint the image at a different location on a different scale. The brightness and saturation adjustment on the other hand uses QColor and repaint each pixel of the image. It then overrides the old image with the new one and updates piantEvent to out paint the new image. Preset would erase all changes, it would set all arguments to its initial states and replace the changed image with the backup of its original version. All values are carefully checked and bounded such that no matter how main adjustments are made there would be no worries of "picture becoming too small" or "colors too bright that is over the RGB limit of (255, 255, 255)". Finally, it enables users to save the adjusted picture in png form by choosing the path and naming the picture by themselves, where a warning will pop up if the saving fails.

4. Contribution

Name	Work
MAO Yiru (118010223)	File System Task Scheduling Exception system Concrete tasks (including the UI design in every task): Login, Browser, Calculator, Calendar, Memo, Monitor (Include display information in main windows), Painter, Main window layout and Time Logo
WU Zhanxin (118010336)	File System Memory Allocation Exception system Concrete tasks (including the UI design in every task): Login, Account (Creation and deletion), Camera, Text Editor, Recorder, Media Player, Basic Logo (User logo, battery and Wi-Fi), Boot Sound
XU Yawei (118010360)	Images (Draw by XU Yawei): background, banners, button icons, bar icon. Photo viewer, Booting interface, Animated buttons, Main Window Layout
Yang Ruichun (118010368)	Exception System Gluttonous Snake, Promotion of save method

Contribution Evaluation:

After collecting the evaluation from every group member, we use the average value:

Name	Contribution Percent
MAO Yiru	33.25%
WU Zhanxin	34.5%
XU Yawei	18.5%
Yang Ruichun	13.75%

5. Reflection

MAO Yiru

At the beginning, I need to construct an overall framework of this fake operating system. But at that time, I was not familiar with the functions offered by Qt to design the UI. Then I search online and find a useful reference book about Qt, named Qt QuickStart. I learned how to create main windows, dialogs, and widgets, make use of the widgets such as push button and line edit. It taught me how to make a connection between the main window and sub-windows.

After that, a simple framework including the login interface, main window, and some sub-windows is built successfully.

Relative path: When I was writing the login part, there are two files: username.txt and password.txt . The username and password entered by the user is compared with that stored in those two text files. At first, I just use the absolute path to read and write those files. However, it's unreasonable to ask the user to change the path in the code. Therefore, the relative path is in need. I firstly tried resources in Qt, but it cannot work for text files. The second method I tried is using QDir::current() function to get the current path, added with the relative path to form an absolute path. This works on my computer. Unfortunately, the QDir::current() works differently on Mac and Windows.

Compatibility on Both Mac and Windows: To make our operating system compatible on both environments, I come up with an idea that if Qt can recognize the system on the computer, thus to compile corresponding codes. I found that the codes between #ifdef Q_OS_OSX and #ifdef Q_OS_WIN are only complied with MacOS and ignored in Windows, vice versa. The problem of the relative path is also solved.

New and Delete: To implement the boot animation, one of my teammates use a widget embedded in a dialog. But every time we closed the project, there is always an exception report. The report shows something wrong with malloc. I remembered in class, the teacher said that the delete pointer only frees the memory space pointed by pointer, but not the memory space occupied by pointer itself. The pointer is still a live variable until it is released. Some trouble will be caused if a pointer is deleted twice. I found that the pointer points to one widget is deleted twice. Then I remove the delete function in the deconstruction, the exception report is disappeared. I think the problem can also be solved if I nullify the pointer after the delete.

Signal and Slots: When I want to show the state of every task on the main window, it caused me some troubles to transfer the message from sub-windows to the main window. Firstly, I thought about if I can use the codes in subwindow.cpp to manipulate the text browser widget in the main window, but failed. Using a global variable is also a way, but I don't want to use too many global variables. Then I searched online and found a good solution, that is to use signal and slot. I set a signal (sender) the sub-window and a slot (accept) in the main window, then use a connect function to link them.

Task Management: Since we fake a memory storage in the operating system, we plan to add a function, that the earliest open task will be terminated automatically when the internal storage is nearly full. In class, the teacher has introduced us to the queue stack structure, with FIFO rule. Something different is the task closed can be removed. When a task is open, I push it back into a vector, it is erased if terminated. Then, when the internal storage reaches the threshold value, the first task will be terminated, like a dequeue operation.

Window Close and Display: When I set the flags for every task, I found that the deconstruction function will not be called by simply close the task. Then I search online and saw other people also have this problem. Someone has solved this problem and offers

“setAttribute(Qt::WA_DeleteOnClose);”, which should be added in the construction function, I tried it and then success. This implementation lays a foundation for monitoring the state of every task. There are also other problems with the sub-window. I open one sub-window (sw1), and then open another sub-window (sw2), the previous window will hide behind the main window, which is not convenient and reasonable. Then I also go online to find help. Fortunately, there is a method that can fix the sub-window upward the main window, thus to solve the problem.

WU Zhanxin

In this project, I encountered many difficulties. First, I need to learn UI by myself. Fortunately, I used to learn UI in XCode and the UI is quite similar between Qt Designer and XCode. To have a brief overview, I read a book called Introduction to Qt that talks about Qt UI details and gives example codes. It is more efficient to learn new knowledge when I read examples. The following are the concrete difficulties and how I overcame them.

Class and struct: In the main window and monitor, all the information for each task is needed. Besides, their information should be connected to the task name. First, I try to create several vectors and each task has its specific location. However, it is hard to remember the location where the task in and hard for teammates to understand. Therefore, I realize that the Class and Struct we learned in class can be used. I create a class for each task and each class has its private variables and functions. The usage of class makes the codes clear and easy to read. Besides, I create a new struct for each task and save all the information into one struct. Therefore, I can combine the memory usage and the task name for memory allocation and main windows.

New type variables and functions: This project needs to use many different types of variables, such as QPixmap, QUrl, and QMessageBox. How to combine them with common type variables and how to use them is a challenge. For each new type variable, I read the concrete explanation from Qt official website. It can give me a brief introduction of each variable. After that, when I need to use them for concrete functions, I use google to search for advice written by others. It is efficient to learn a new type of variable from concrete examples from others.

Deletion: In the Media Player part, if the user closes the window when a media is playing, a segmentation fault will occur. To solve the problem, I first try to close the Player when the window is closed. However, the segmentation fault still existed. Therefore, I looked over the information about Bad Access Exception and finally found that I created a new Player without deleting it. This experience reminds me to pay attention to the usage of new. I checked all the new in the project and fix the segmentation fault, so that there is no error when users close windows of Camera or Media Player.

Extern variables: To work as a team and make codes clearly, our project has nearly fifteen

cpp files. Some variables may be used by different files and it is easy to include headers mutually for these files. Finally, I create a new header file and a new cpp file that is mainly for extern variables, so that the errors can be avoided.

Difference between MAC OS and Windows: This project is decided for both MAC OS and Windows computers. However, there are some incompatible situations in these two OS. First, the same UI codes have different layout. Second, the way to get file information such as author and creation time is different. Third, the definition of absolutePath function is different. At first, my group member and I try to find a new way that is suitable for both two systems, but we failed. Later, we realize we can write different codes for different OS. My program will first check the OS type and then designed a specific layout and used different methods based on its OS. Therefore, the project can work both in MAC and Windows.

Update timely (battery and Wi-Fi): Some information in the fake OS is needed to be updated per second, such as Wi-Fi connection and battery. First, I try to find a function or class that can update information timely. However, I cannot find such code and decided to connect the time function and update function. Finally, I initialize the battery and the Wi-Fi and the time function will check the Wi-Fi per second and decrease the battery automatically. Besides, the logo for battery and Wi-Fi will change respectively.

XU Yawei

I would like to divide my reflections by different classes I implemented.

(1) the Photo class (used for photo viewer)

When writing the photo viewer, I first intended to store all status in private variables such that during preset all I need to do is to set all of them to the initial value (scale 1.00, others 0) and update paint event. However, when adjusting brightness and saturation all values need to be bounded to (0, 255) due to the color display theory adopted by all computer screens. Even though I intended to change RGB value by 30 in brightness adjustment and saturation value by 10 in saturation adjustment, there exists time when values have reached its limit and ceased changing. I can of course, save the old value and do subtraction and mark down the exact changes, but QColor do the color operation pixel by pixel, which means I have to save an entire pixel map of delta RGB and HSL, which takes the same storage of saving another image. Therefore, I changed my code such that during loading in the image I actually saved it to two QImage objects, one for display and adjustment, and the other for recovery. Then when Preset is called all I need to do is to replace the current image with the backup one, set other value to initial and call for an update of the paintEvent function.

(2) The Opening class (used as the booting interface)

Since I was in charge of most of the works related to image display and animation. I always try find something in common such that to decrease my work load. When I was writing the animation for booting interface my first attempt was to use the same methods adopted in Photo

Viewer i.e. changing some private intervals and calls an update of the paintEvent function. To make them an automatic animation, all I need is to introduce QTimer and set a loop timer that change the image slightly in every 50 millisecond or so. Yet I was quite satisfied with the performance of the QAnimation class applied in MyPushButton class. It provides easing curves that makes the animation more vivid instead of performing in uniform speed. In fact, I think the bouncing movement preformed when a customized push button is clicked is what I want the loading dots to do during booting up. Therefore, instead of drawing pixel maps in the booting interface I introduced several customized push buttons as animation units. They are not connected to respond any action, but only to perform animated movements according to the loop countdown of QTimers.

The remaining problem is that since these are MyPushButton objects they respond more like a push button objects: First, the background changes colors when cursor hovers over it; Second, the cursor changes its shape (to the shape that intents user to click) when hovering over them. These effects become draw backs when using them as animation units. To solve that, I added a Boolean parameter to MyPushButton class named "AsButton". This is used to judges whether this object is used as a pushbutton or an animation unit. I also rewrite the constructor of MyPushButton class, giving it an if statement and stripping all button-like features if AsButton is set to be false. Then the constructor would initialize an object that are only able to display an image and preform animations, nothing more. This decision turned out more important then I thought when later on I begin to add tips widgets to MyPushButton class. I can shrug off performance such as changing in background color when cursor hovers as "playful functions", but if a widget bounce out every time the cursor passes the animated pictures that would cause huge confusion to the operating users.

(3) MyPushButton class (used for any push button that has a costomized image outlook)

When a cursor hovers over a customized pushbutton, three changes happens:

- The cursor changes its shape.
- The pushbutton brightens its background slightly.
- A Widget with tips about this application appears right of the pushbutton.

Although they seem to response to the same thing (cursor hover), the implementation is completely different.

The change of background can be simply done by adding a hover condition in the style sheet, a basic operation for most layout designs. The change of cursor was done by applying QCursor class in MyPushButtonClass. Then when cursor enters a customized pushbutton of MyPushButton class it would change automatically. This is much better than applying QCursor in MainWindow class and calculates at which spot it should change its outlook. The display of tip widget was a much more difficult story. The tips widget needs to appear when cursor hovers on the button, and hide itself when the cursor leaves. Yet for event functions there is no such events as "hover" or "not hover". I finally chose enterEvent and leftEvent for implementing the

detection. I override these two events such that when cursor “enters” the pushbutton, a tips widget would appear; the same widget disappears when the cursor “left” the pushbutton.

Then I was faced with the problem of moving the tips window to the correct location, which should be almost the same position as the pushbutton. I tried to use the maptoGlobal function I utilized when writing the Photo Viewer and capture the cursor’s position, but when this function was applied to MyPushButton class, who inherits from QPushButton class instead of QWidget class, it operates differently and the tips windows were not placed in the location I desire. Then I realized I moved each pushbutton to a certain location using the move (int x, int y) function. I could use the same function again with the same location to the widget initializing main window if I set the widget as a public variable. A better approach would be setting the x and y position as a private variable, and override the move function of MyPushButton class so after moving the button to certain location it also stores that location for moving tips widget.

Since the pushbutton has a considerable size (8*8 cm with 72pixels per cm), a problem is possible that the widget overlaps with its pushbutton. When the widget appears, cursor is detected inside the widget and outside the pushbutton, thus the widget should then be hidden; and then the cursor would return onto the pushbutton which brought the widget back. This would result in tips widget flashing on and off, finally crushing the system. Thus, the widget needs to be position at location that holds a small distance apart from its pushbutton.

When testing the tips widget on an IOS system another problem occurred: The quit button was on the upper left side of the window very close to the calendar pushbutton. It is possible for a user to shut down the system and after that move to the calendar button and display its tips widget. Since the widget is written in MyPushButton class (thus main window is not set to be its parent) the tips widget would keep to be there even after the system is shut off. To solve that I added a QWidget parameter to the input of MyPushButton’s constructor. Then I can pass main window inside and set it to be tips widget’s parent. As a result, all tips widget would automatically close itself when main window is closed.

YANG Ruichun

Implementation of Gluttonous Snake

I encountered most difficulties during the implementation of the video game “Gluttonous Snake”. When I was doing the first version of the game, the first problem I met is how to display the game interface. The first idea is to use the console to display the game by using gotoxy function to draw points on certain coordinates and draw the interface structure. However, when I add the game into our project, I found that console cannot pop up in the qt-widget-application-based project. After searching for the reasons, I learnt that qt application cannot open the console by default and adding “CONFIG += console” can solve the problem. However, it is still a problem that the open of console is quite slow in the qt application and the console display screen is too simple and crude. At the moment, my teammate mentioned that the game can be written in the way of qt widgets. After learning several qt classes, I chose QWidget class to

inherit and did override on the function paintEvent and keyPressEvent to build the connection between users' input and the change of game screen and thus I had to rewrite the whole structure of codes. During this difficult process of implementing the Gluttonous Snake, the inheritance that I learnt in class really helped me understand how to use the two special event function of the original QWidget class in my game class, but more useful things to the project were learnt in the Qt application class tutorials, such as QFileDialog, QPainter and QMessageBox.

However, the coding of this small video game was not over. The application raised s new problem that the game was still running after it was closed. After several testing I realized that clicking the closed button just hided the game subwidget instead of deleting it, which meant that the destruction function did not work when clicking the cross button or the "quit" button in the message box when game is over. The destruction functions of subwidgets only works when the whole parent window is closed. I searched for the solution on google and found that I can set attribution of the game widget in its construction function, so the destruction function will work to make the pointer object of my game class free just when the game window is closed.

Debugs with Qt error message

Debugs was another challenge for me since I was new to Qt widget tools and it was easy for me to make many mistakes on the usage of various qt functions. One experience that really impressed me is that when I was testing my painter, I found it strange that though the program was successfully compiled, the picture did not appear and the program kept telling me "painter not active" and returned a number "2". The answers in google told me that it was because the QPainter object did not connect to a QPainteDevice object. However, I repeatedly checked my code and made sure the QPainter indeed connected to a painter device. After searching and testing the error for a long time, an idea suddenly hit me that I focused my attention on the return number "2". I checked the source code of QPainter and found that "0x02" return value is correspond to a certain type of QPaintDevice: pixmap. Finally, I located a pixmap which was not correctly instantiated due to the argument QSize() (which should be size()). The experience finally taught me that while using an unfamiliar tool, it is very important to understand every information conveyed by the error massage, which is a great help to debug.

Learning new tools

The Qt widget application tool used to build our project is totally unfamiliar to me. By learning the Qt Creator Manual, I firstly constructed a simple Qt widget based application with QMainwindow class and learnt how to connect the ui button, signals and slots. However, using ui design interface was not enough because its layouts were simple and crude. After searching and learning more tutorials and samples, I gradually got used to doing the layouts with codes instead of dragging buttons on the ui design interface. Moreover, I gradually learnt various classes including QTimer, QPainter, QMessagebox, QToolbar and QFileDialog to satisfy many different needs such as saving files in the single application (ie. photo), updating game interface in intervals and giving error prompts as a real exception system does. Although Qt widget tool is

difficult for new users to learn, the learning experience enhanced my ability on using new programming tools and showed me samples of doing GUI designs and functions that implement the interaction between programs and users.

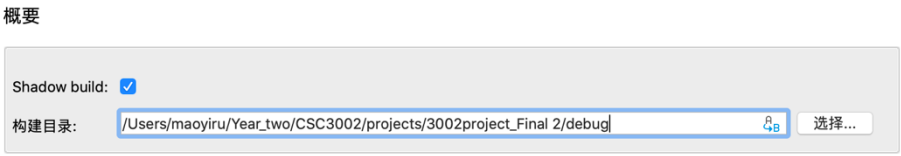
6. Usage:

MacOS is more recommended to use our Operating System, since the layout is tidier and more elegant on MacOS.

To whom may test our code:

Step1. Click .pro file to open Qt Creator.

Step2. Set the build path as the debug file, which is provided in the file.



Step3. Click build button.

Step4. There is a login interface, we offer 7 initial accounts, you can use the account to enter this operating system. The secrete password for deleting account is CSC3002.

Username:1

Password: 1

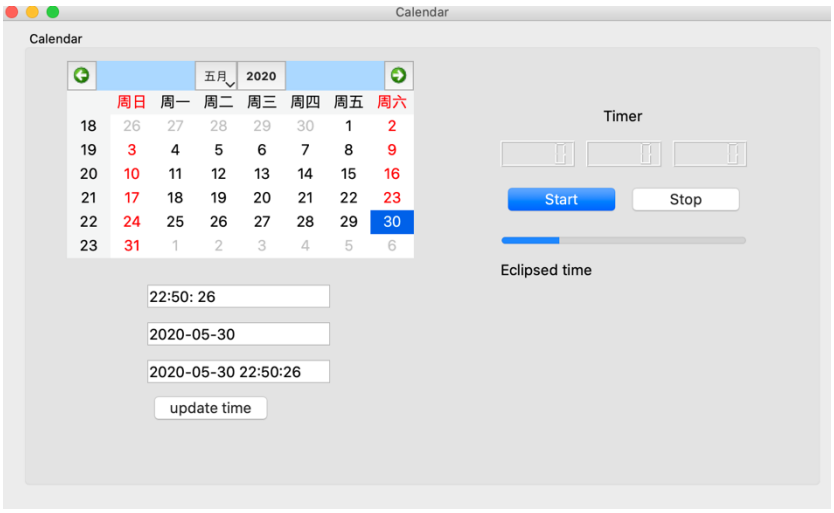
Appendix: User Manual

Content:

1) Calendar

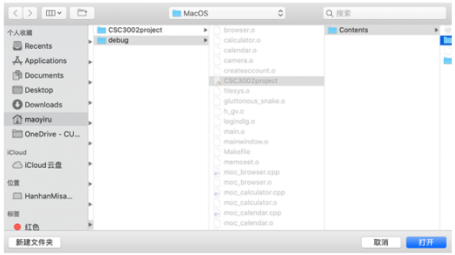
It includes a calendar, current time and a timer.

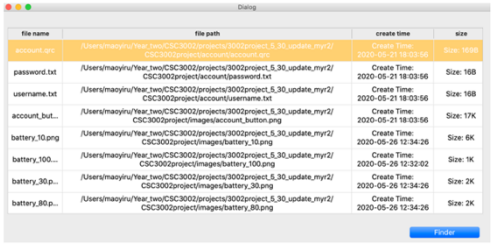
The timer starts when the user clicks the start button, stops when the user clicks the stop button, the eclipsed time will then be shown.

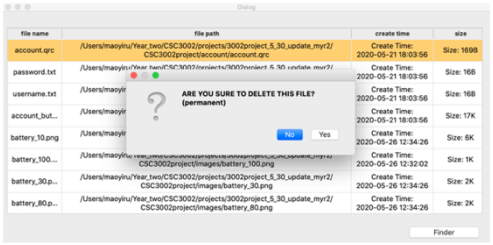


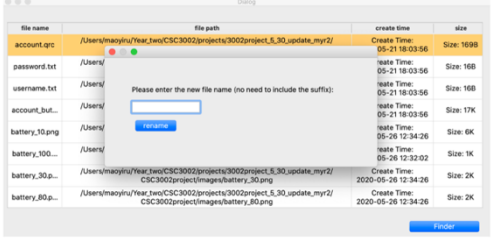
2) File System

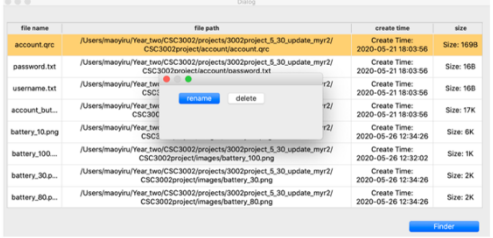
1. Click the finder button to choose the file to be open.
2. Double click the file in the table to rename or delete.
3. Users can input the new name of the file.
4. The warning is shown.
5. Click the "refresh" to refresh the table.





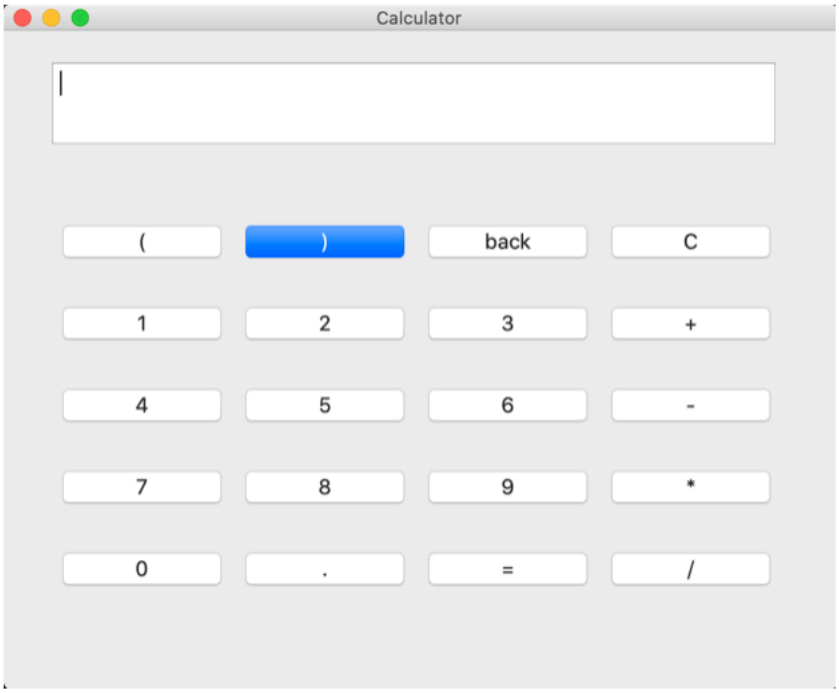






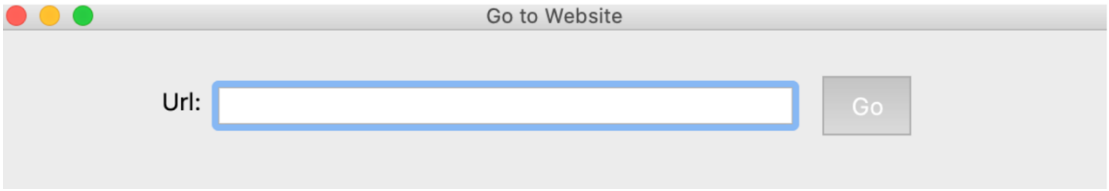
3) Calculator

It is a simple calculator. Click “C” to clear



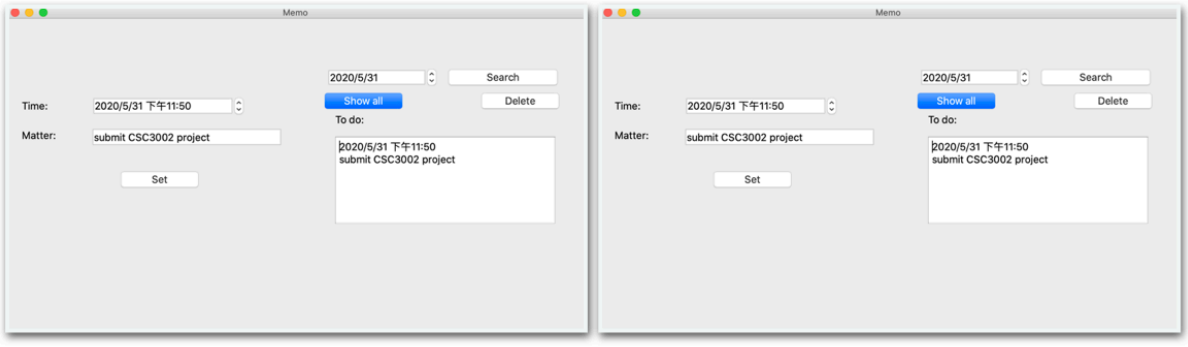
4) Browser:

1. Input URL.
2. Click “Go” to visit the website



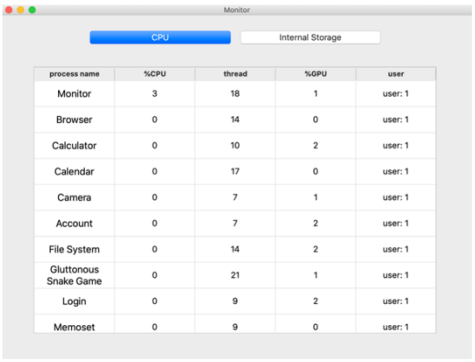
5) Memo:

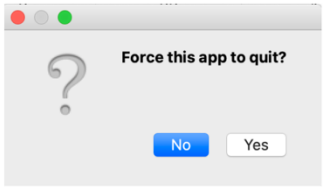
1. Choose time and input matter, click set to store the memo.
2. Click “Search” to show the memo set on a specific day.
3. Click “Show all” to show all the memos after the current time.
4. Click “Delete” to delete all the memos before the current time.

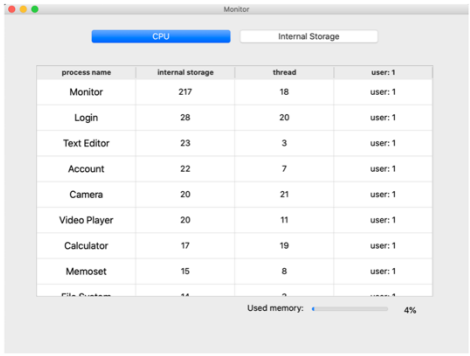


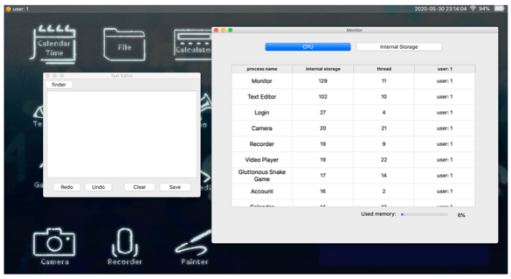
6) Monitor:

1. Click CPU to monitor the CPU condition.
2. Double click the task to quit.
3. Click Storage to monitor the internal storage condition.



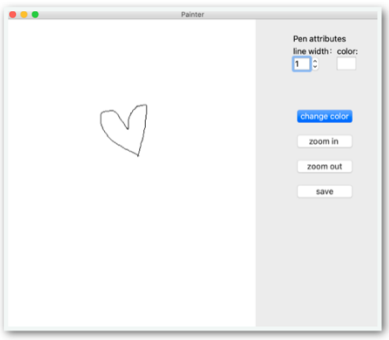


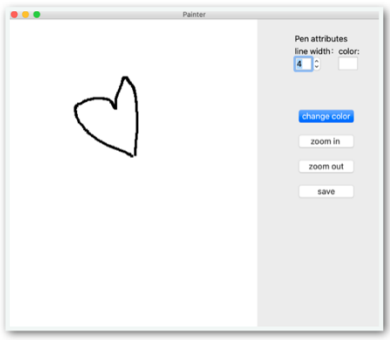


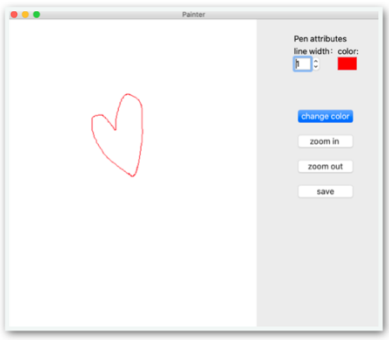


7) Painter:

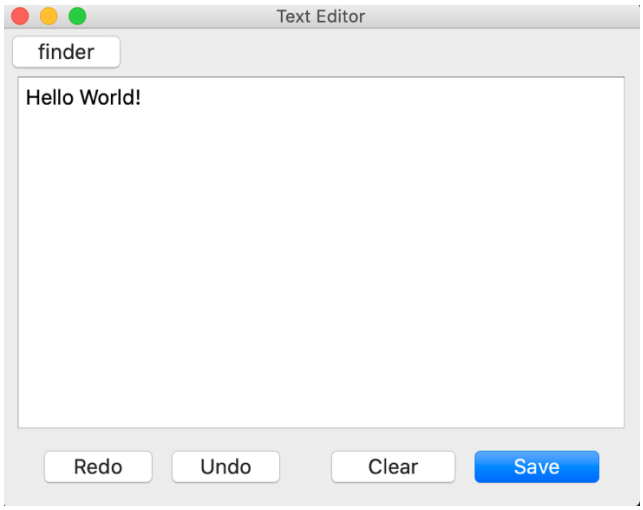
1. Use spin box to set the pen width.
2. Click “change color” to change the pen color.
3. Click “save” to save the picture into PNG form.
4. Click “zoom in” to zoom in the canvas.
5. Click “zoom out” to zoom out the canvas.



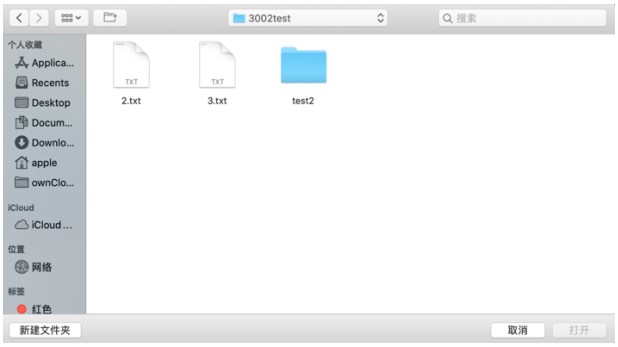




8) Text Editor:



1. Click the finder button to find the text file you want to edit. When the text editor is open, the initial text file is test.txt in the project folder.



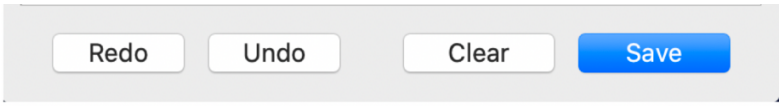
2. Edit the file.

Click the undo button to undo last operation.

Click the redo button to redo last operation.

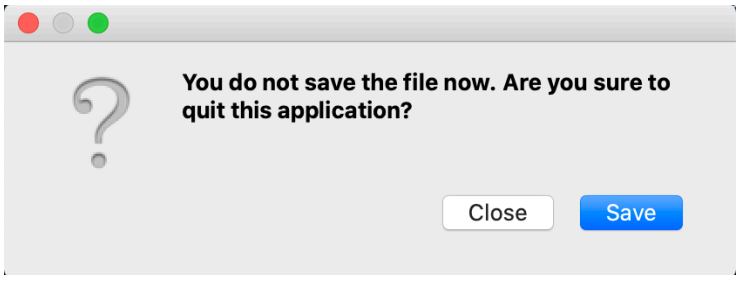
Click the save button to save the file.

Click the clear button to clear the file.



3. Close the window

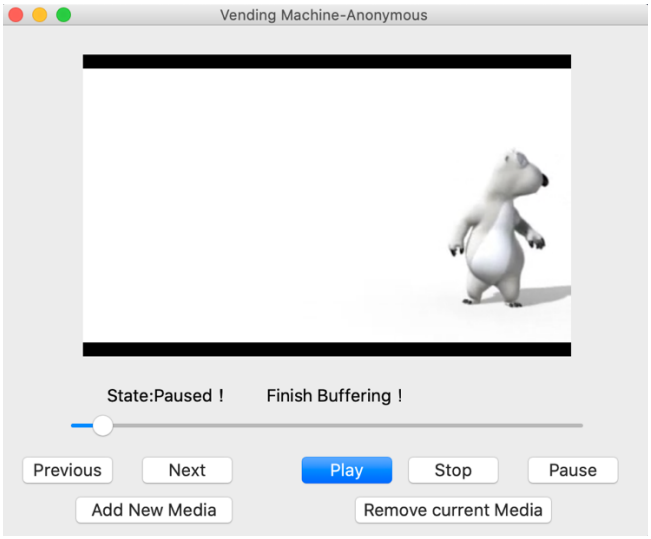
If the content is changed and the user do not save the file, the following question will show up.



Click the save button to save the file.

Click the close button to close the window without saving the file.

9) Media Player:



The Media Player initially has a video and a song. The slider expresses the pace of current Media. The State and the State change are under the media widget. The Media name and its author are in the title of this window. When the Media Player is open, the playlist is initialized to have a video and a song.

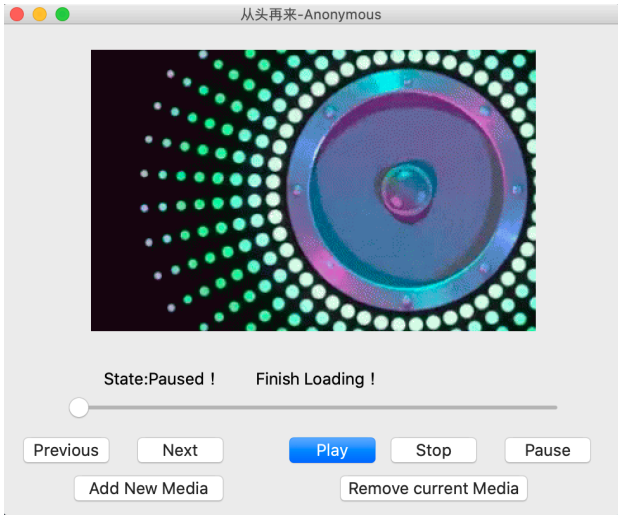
Click the play button to play the media.

Click the stop button to stop playing the media.

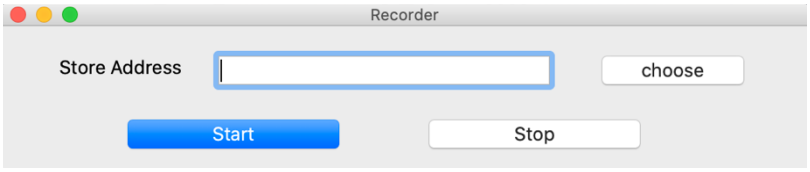
Click the pause button to pause the media.

Click the previous button to play the previous media in the playlist.

Click the next button to play the next media in the playlist.



10) Recorder

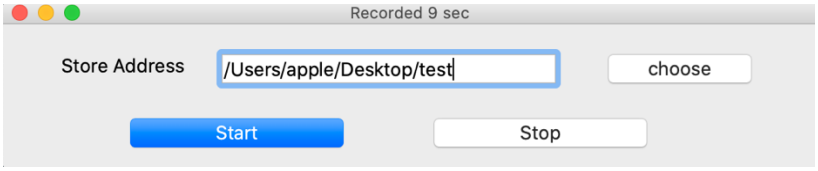


Click the choose button to find the path where you want to save the sound file. Users can also enter the path by typing.

The file is saved automatically in WAV format when the stop button is clicked.

Click the start button to start to record.

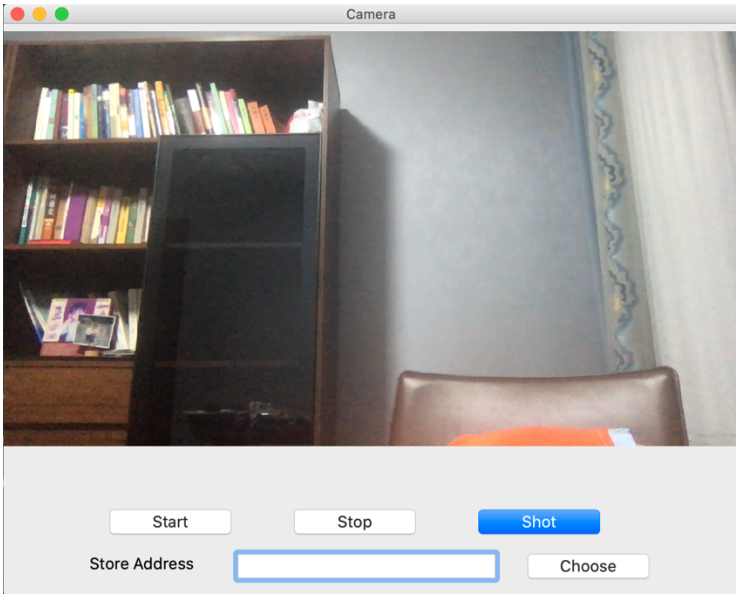
Click the stop button to stop recording and save the file.



If the user does not choose a path, the following warning will show up.

11) Camera

When the camera window is open, the camera is on.



The file is saved automatically format when the stop button is clicked.

Click the choose button to find the path where you want to save the sound file. User can also enter the path by typing.

Hint: the store address should include suffix, like ".mp4".

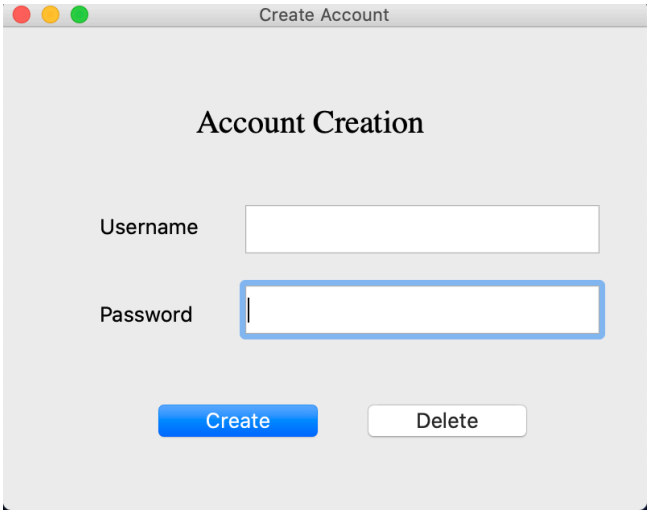
Click the shot button to start to take a picture.

Click the start button to start to take the video.

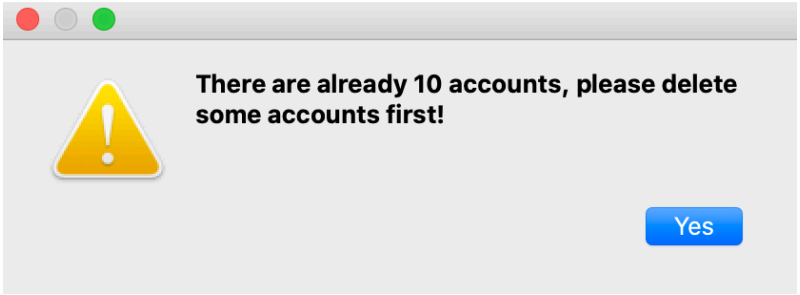
Click the stop button to stop recording and save the file. If the user does not choose a path, the following warning will show up.

12) Account

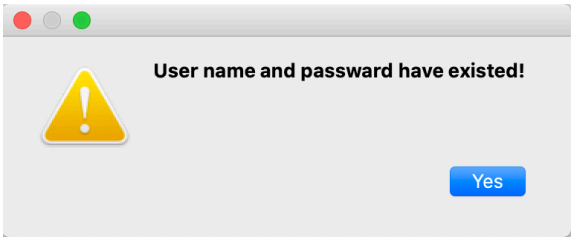
- Create Account



1. User can also enter the username and password name and click the create button to create an account. The OS will allow at most ten accounts.
2. If there are already ten accounts and the user want to create more account, the following warning will show up.

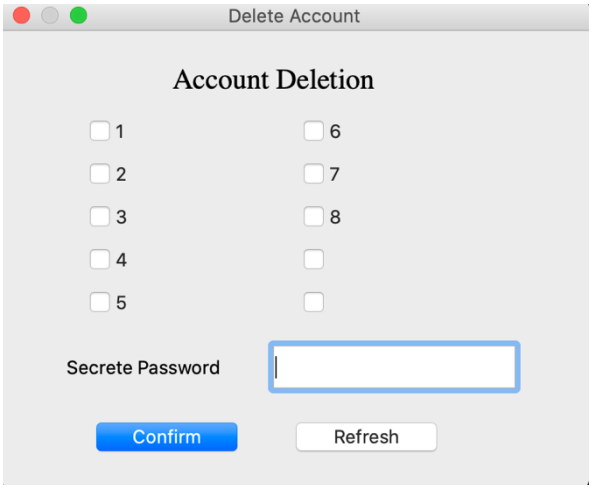


3. If the account that user want to create has existed, the following warning will show up.



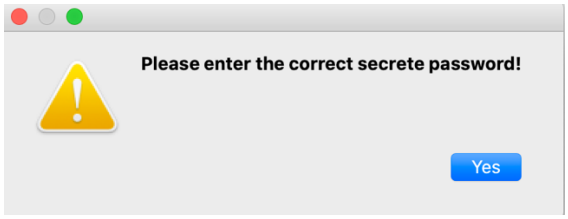
Click the delete button to delete account.

- Delete Account

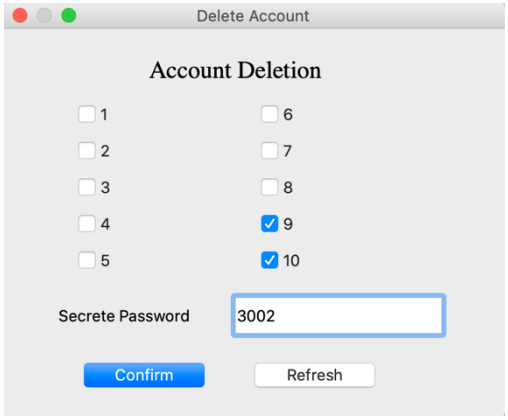


User need to choose the accounts that he or she want to delete and enter the password. Then click the confirm button.

If the password is wrong, the following warning will show up.

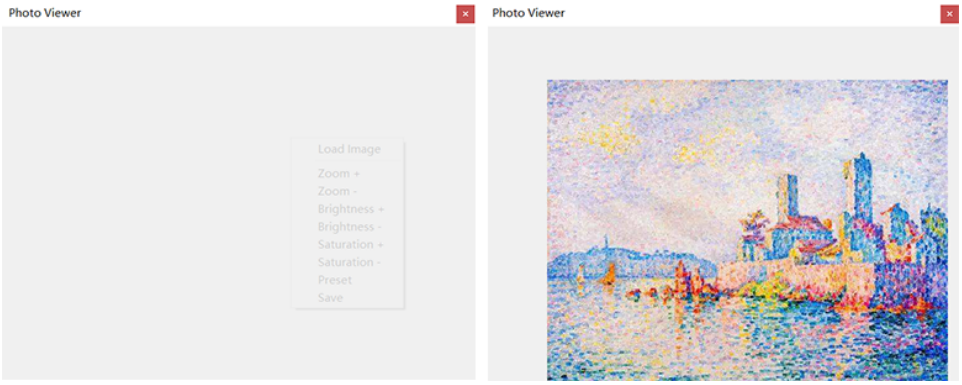


The correct way to delete account is in the following. The secret password is 3002.



13) Photo Viewer

1. Right click to open context menu. Choose "Load Image" and pick the image you would like to view from your computer.
2. After loaded it in, press & drag to move the image, use wheel to zoom in and out. Right click for more adjustments in the context bar. All of them can be applied multiple times.
3. If you are satisfied with your change, right click and choose "Save" to save your adjusted image. If not, choose "Preset" to change it back to the original state.



14) Gluttonous Snake

1. The left area of the game interface is the game zone while the right area display the real-time score and operation instructions.
2. When the game is over, the message box will appear to tell you the score. Click "Start New" to play a new round; click "quit" to leave the game.