

模擬餐飲業 POS 系統

1112_2162_物件導向程式設計(資管系一甲)_資管系一甲

作業：week12_pos 系統

學生：楊士霆

學號：C111118223

班級：資管一乙

指導老師：黃承龍 老師

指導助教：塗浚伸 助教

內容

壹、 前言	3
一、 名稱	3
二、 動機	3
貳、 設計架構	3
一、 流程圖	3
二、 畫面架構圖	3
參、 成果	4
一、 框架	4
(1) ingrediate.java	4
(2) Commodity.java	5
二、 成果圖片與介紹	8
肆、 延伸	13
一、 顧客系統	13
二、 帶位訂位系統	13
三、 收銀機系統	13
四、 銷售分析	13
五、 隨機事件	13
六、 銷售策略	13
伍、 結論	13
一、 差異	13
二、 困境	13

圖表

圖 1 遊戲流程圖	3
圖 2 SHOP 架構圖	4
圖 3 遊戲首頁	9
圖 4 商城介面	9
圖 5 食材購買介面	10
圖 6 已購買食譜介面	10
圖 7 食譜購買介面	11
圖 8 購買紀錄	11
圖 9 統計數據	12
圖 10 購買紀錄檔案	12

壹、前言

一、名稱

根據下述動機，決定將本專題取名【模擬餐飲業 POS 系統】。

二、動機

由於本身期望透過此專案，更加了解 POS 系統的架構，同時也希望讓使用者能夠體驗餐飲業的角度，操作 POS 系統來應對餐飲流程，包含原物料控管、帶位訂位、訂單、送餐、銷售控管等……

為達成期望，因此選定模擬餐飲業作為本次主題。

貳、設計架構

一、流程圖

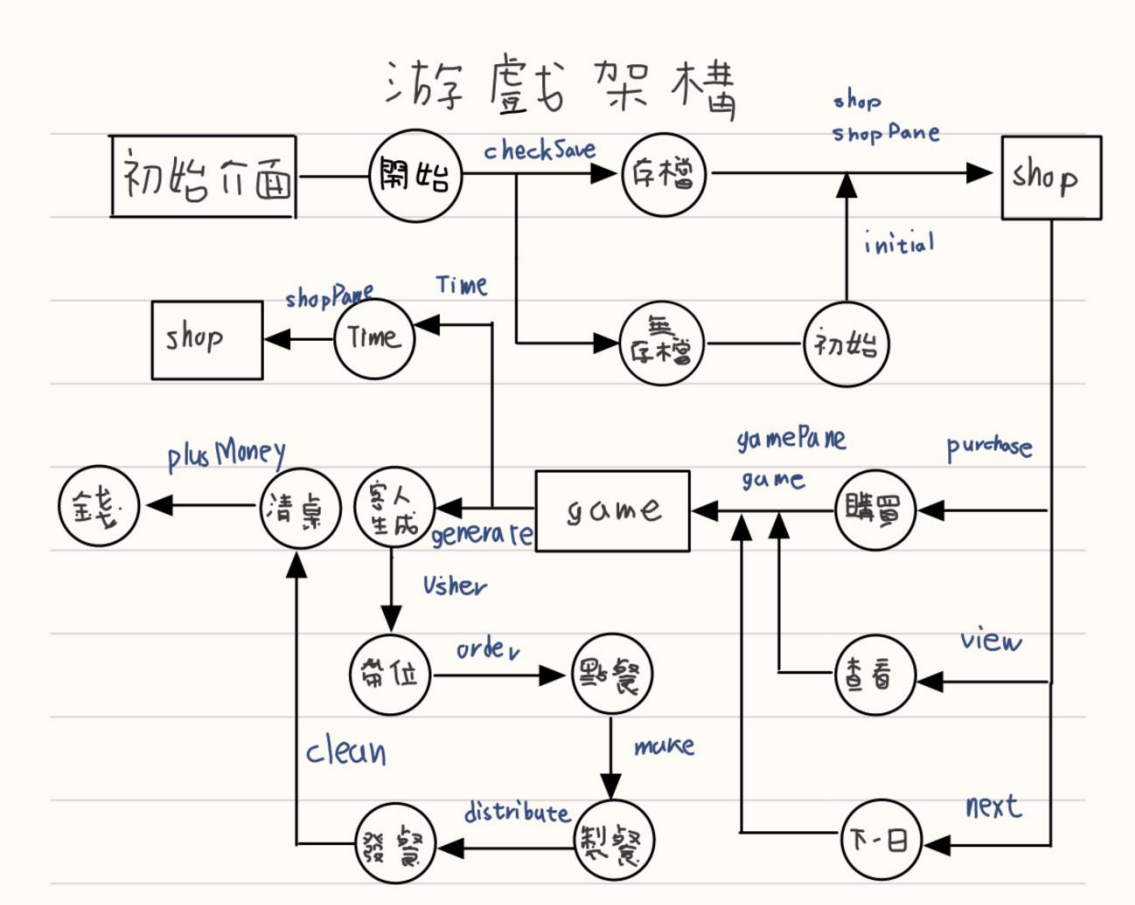


圖 1 遊戲流程圖
(資料來源：本人繪製)

二、畫面架構圖

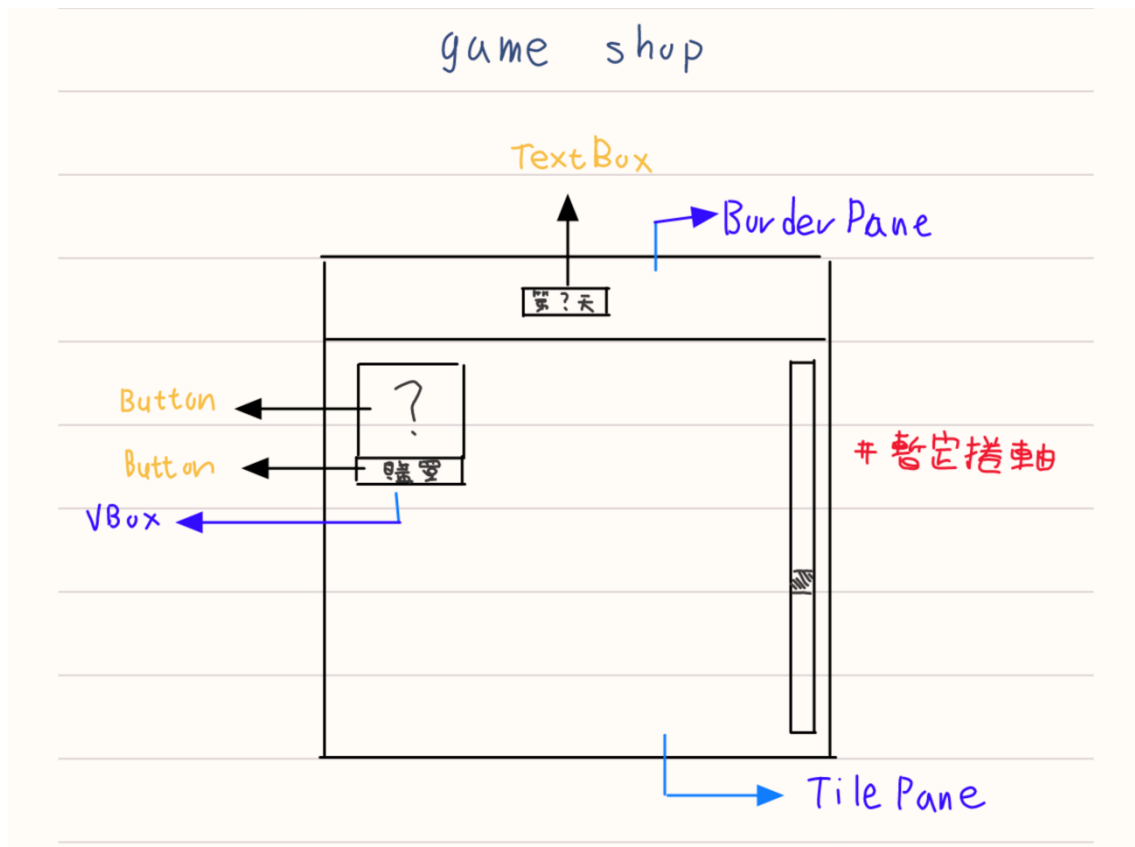


圖 2 SHOP 架構圖
(資料來源：本人繪製)

參、成果

一、框架

本次設計將產品分為原物料(ingrediate.java)與菜單(Commodity.java)，使用者必須花錢購買菜單才能習得製作方法(倘若尚未購買，即使使用正確物料及方法，也無法製作成功)，且必須購買原物料才能著手製作。

(1) ingrediate.java

欄位分為：

- String id(食材編號)
- String name(食材名稱)
- String image(照片名稱)
- Int cost(購買成本)
- Int nowStock(目前庫存)
- String describe(食材描述)
- String origin(原產地)

主要方法：

- `public void SetNowStock(int nowStock)`
用於更新購買後的庫存
- `public String[] getAllDatas()`
回傳給商城點擊食材後產生的介面所需的相關資料
- `public String getAllDatasForSave()`
回傳存檔所需的資料

(2) `Commodity.java`

欄位名稱：

- `String id`(產品編號)
- `String name`(產品名稱)
- `String image`(照片名稱)
- `Int cost`(製作所需總成本)
- `Int price`(可售價格)
- `List<String> ingrediates`(製作所需食材)
- `String describe`(商品描述)

主要方法：

- `String[] getAllDatas()`
回傳給商城點擊食材後產生的介面所需的相關資料
- `String getAllDatasForSave()`
回傳存檔所需的資料

並在整體程式碼的流程分為 `GameManager.java` 以及 `PaneManager.java`，前者用於處理遊戲相關及存檔讀檔，後者用於處理所有畫面呈現與建立。

(1) `GameManager.java`

欄位名稱：

- `PaneManager pm`
用於與 `PaneManager` 互相調用
- `String day`
用於紀錄目前遊戲天數

- String money;
用於紀錄目前財產
- IngrediateList iList
用於存儲商城可購買食材的 List
- CommodityList cList
用於存儲商城可購買食譜的 List，倘若被購買過一次，則不會出現在此 list 中
- CommodityList mcList
用於存儲已購買的食譜
- String[][] datas
用於紀錄至今為止，各食材被購買的相關數據

主要方法：

- public void PlusDay()
用於更新日期
- public void Save()
用於將所有數據更新、存儲在 save.txt 中
- public void PurchaseCookBook(String id)
用於購買食譜後的數據處理
- public List<Button> UpdateShopMenu(int index)
回傳給 SHOP 刷新所需的 Buttons(購物選項)，分為三種 mode，可分別回傳食材、食譜、已購食譜三種 Buttons
- public String getALLRecord()
回傳過去所有購買紀錄
- public String getRecordData()
回傳各食材被購買的詳細資料紀錄

- `public void initial()`
用於讀檔與初始化各種數據
- `public void record(String new_record, String[] data)`
用於購買後的相關紀錄數據儲存

(2) `PaneManager.java`

欄位名稱：

- `GameManager gm`
用於與 `GameManager` 互相調用
- `Pane root;`
最基礎的介面
- `Pane allPane`
第二基礎介面
- `VBox mainPane`
主要介面
- `VBox gamePane`
餐飲遊戲介面(暫被作為分析介面)
- `TilePane shopMenu;`
商城購買介面
- `Label nowDay;`
顯示天數
- `Label nowMoney`
顯示擁有金額
- `StackPane vicePane`
副介面，目前用於存放單一食材購買畫面

主要方法：

- `public void GamePane()`
建置遊戲介面

- `private void MainPane()`
建置商城介面
- `public void SetGameManager(GameManager gm)`
設定 GameManager，以利後續相互調用
- `private void StartBorderPane()`
設定遊戲首頁
- `public void showCommodity(String[] datas)`
設定點擊商城中食譜後產生的購買介面
- `public void showCommodityData(String[] datas)`
設定點擊商城中已購買食譜後產生的製作方法介面
- `public void showIngrediateData(String[] datas)`
設定點擊商城中食材後產生的購買介面
- `public void updateIngrediateBuyingNum(Label l, int num)`
更新食材的購買數量
- `public void updateShopPane(List<Button> menu, String day, String money)`
當選擇食材、食譜、已購買食譜後，更新商城中的商品
選項方法

二、成果圖片與介紹

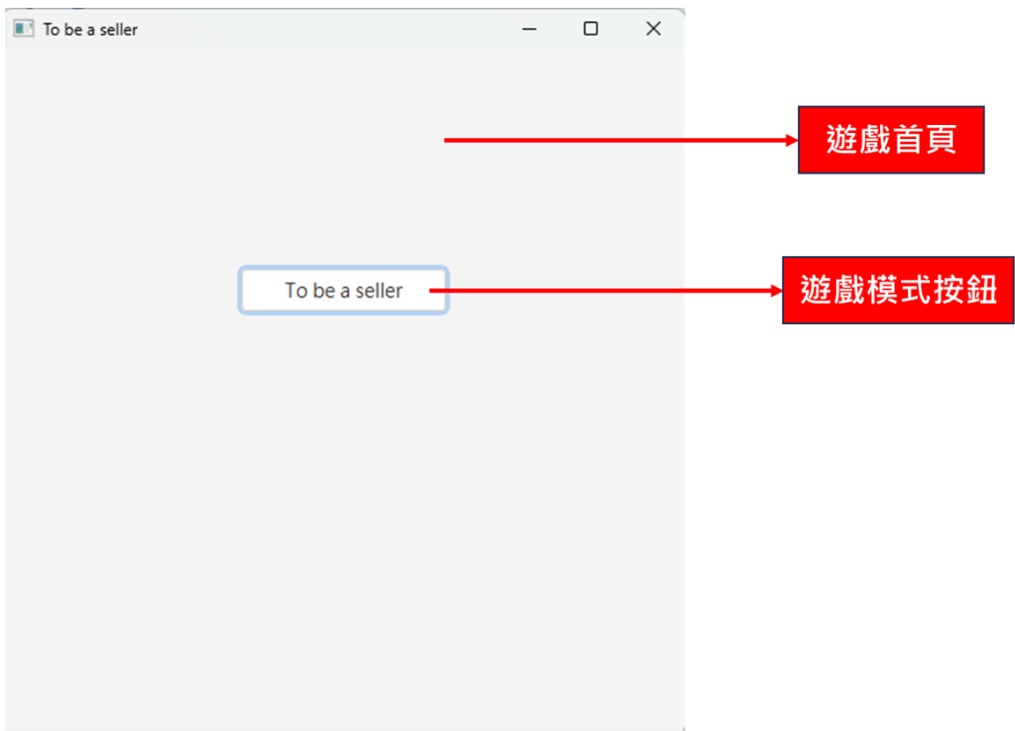


圖 3 遊戲首頁
(資料來源：本人繪製)

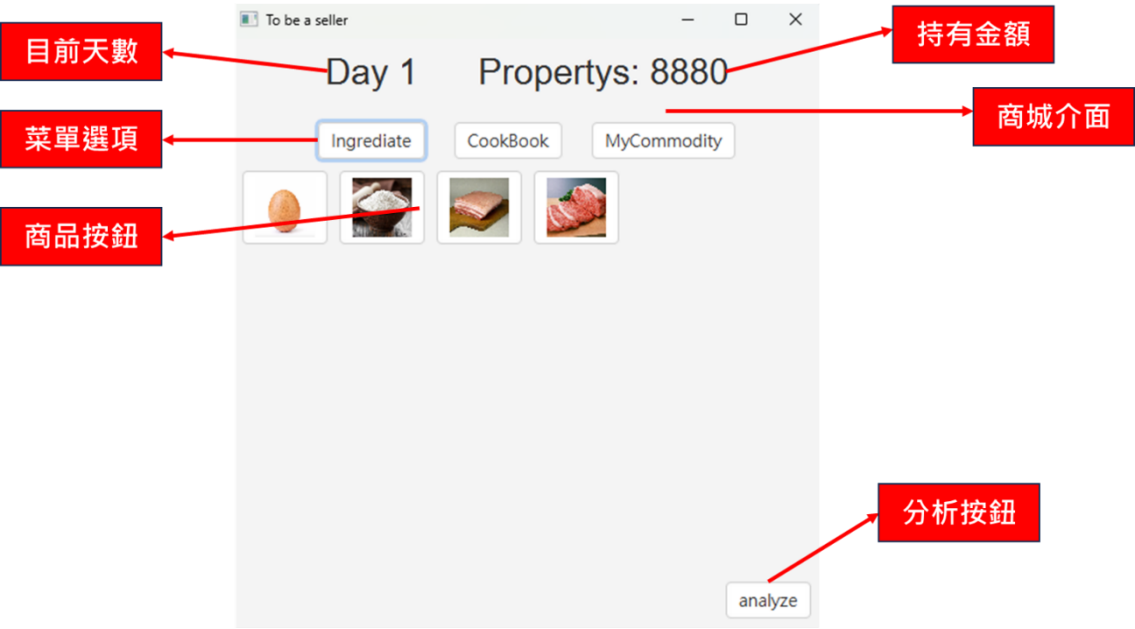


圖 4 商城介面
(資料來源：本人繪製)

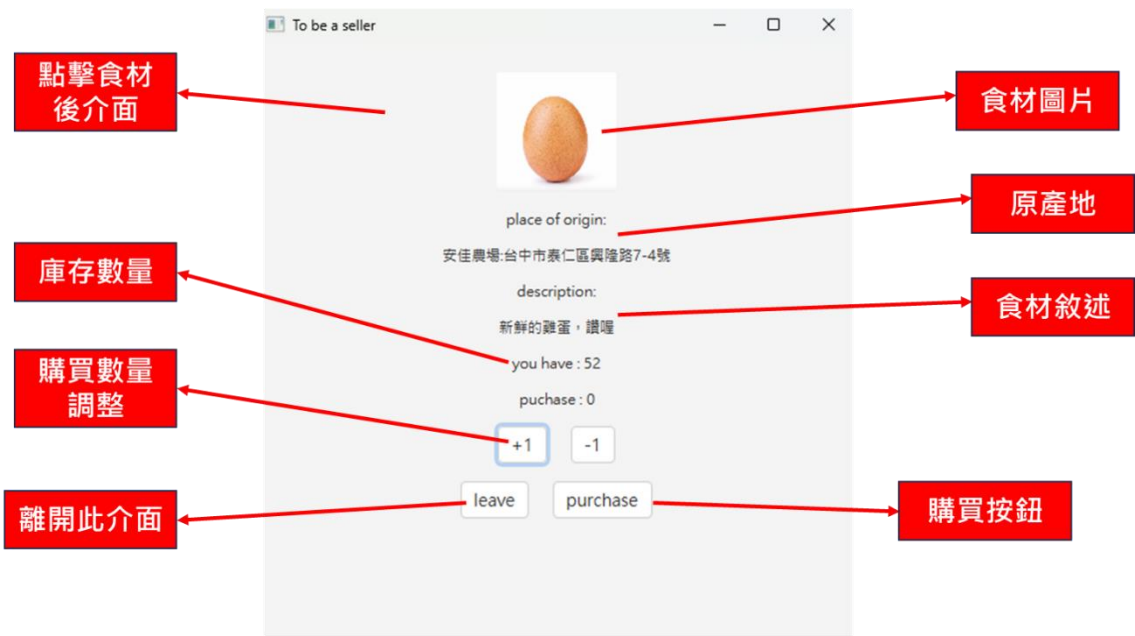


圖 5 食材購買介面
(資料來源：本人繪製)

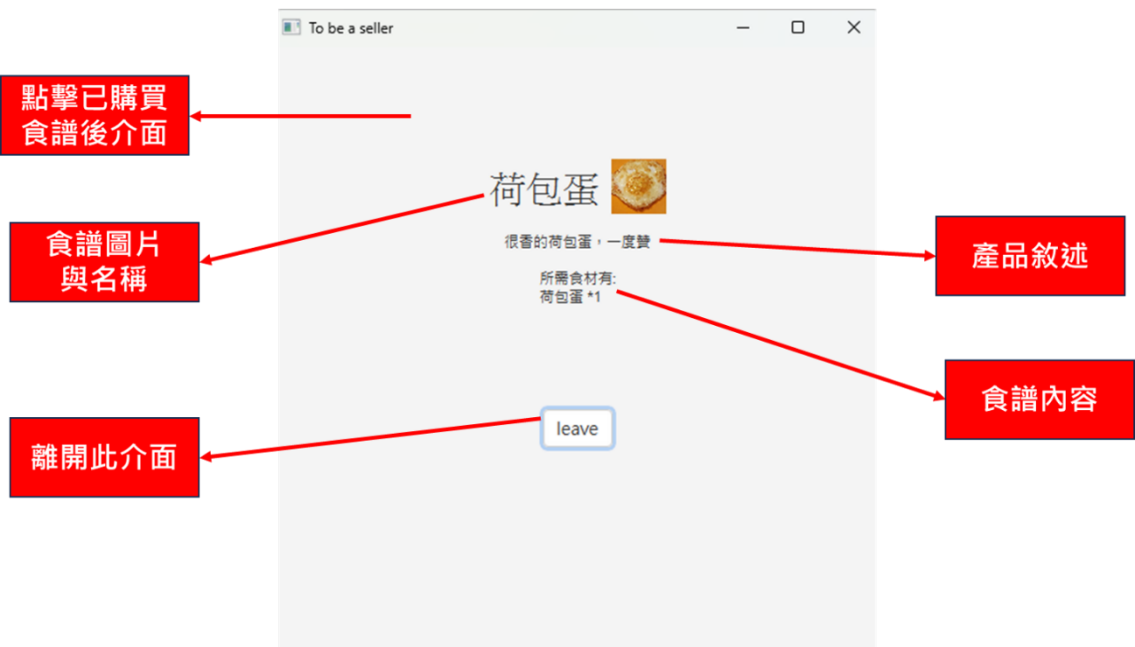


圖 6 已購買食譜介面
(資料來源：本人繪製)

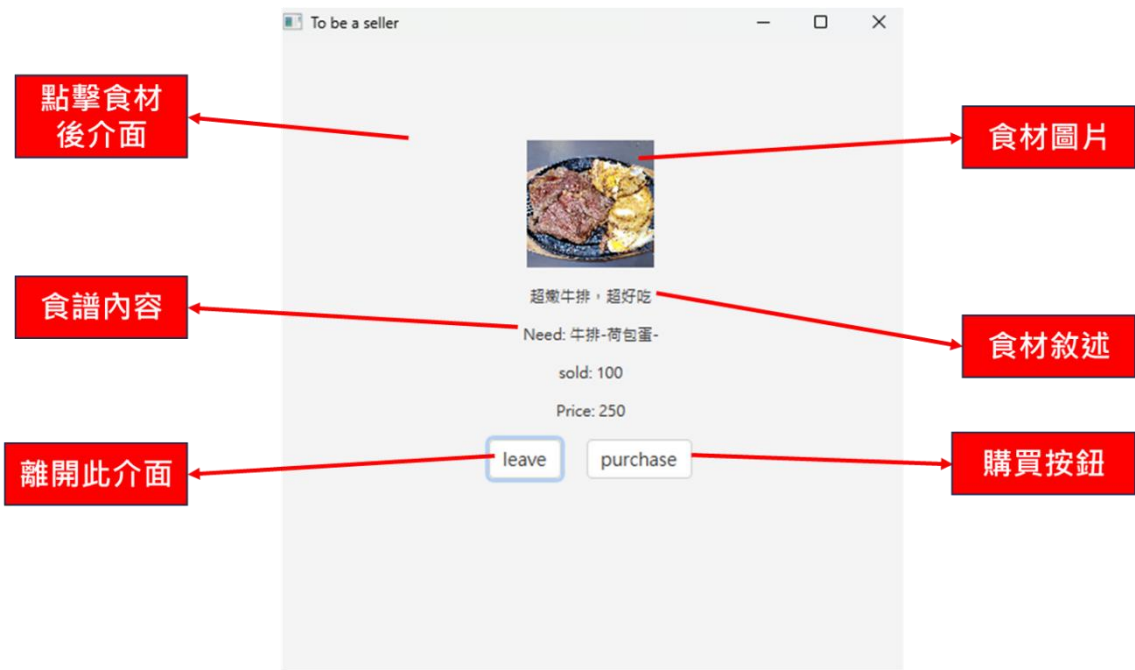


圖 7 食譜購買介面
(資料來源：本人繪製)

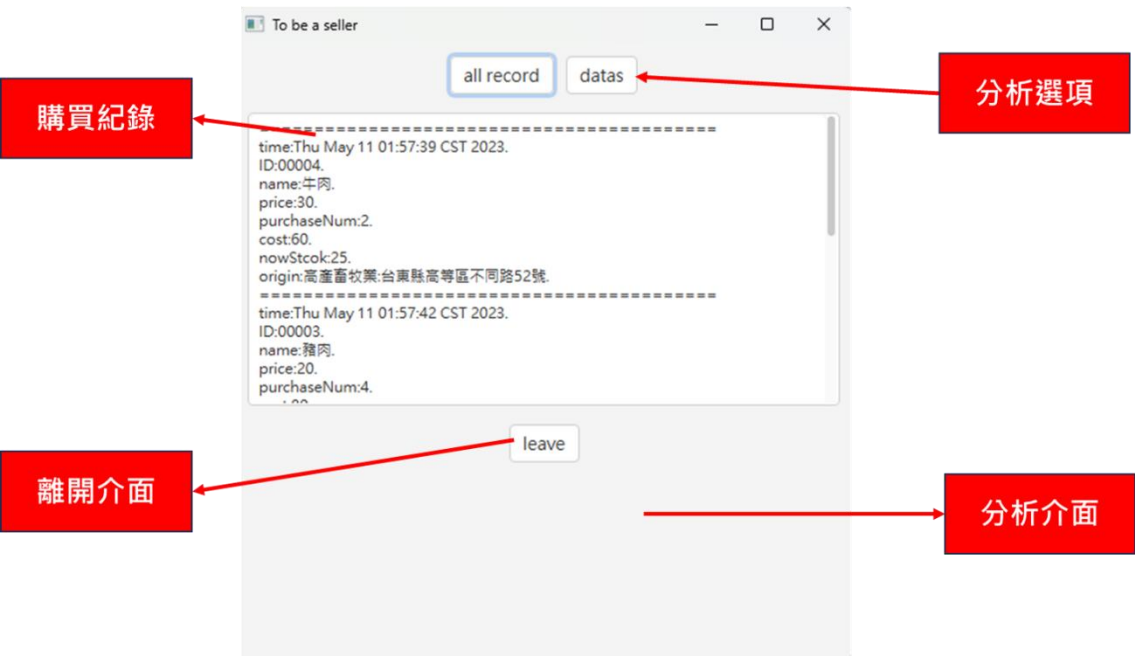


圖 8 購買紀錄
(資料來源：本人繪製)

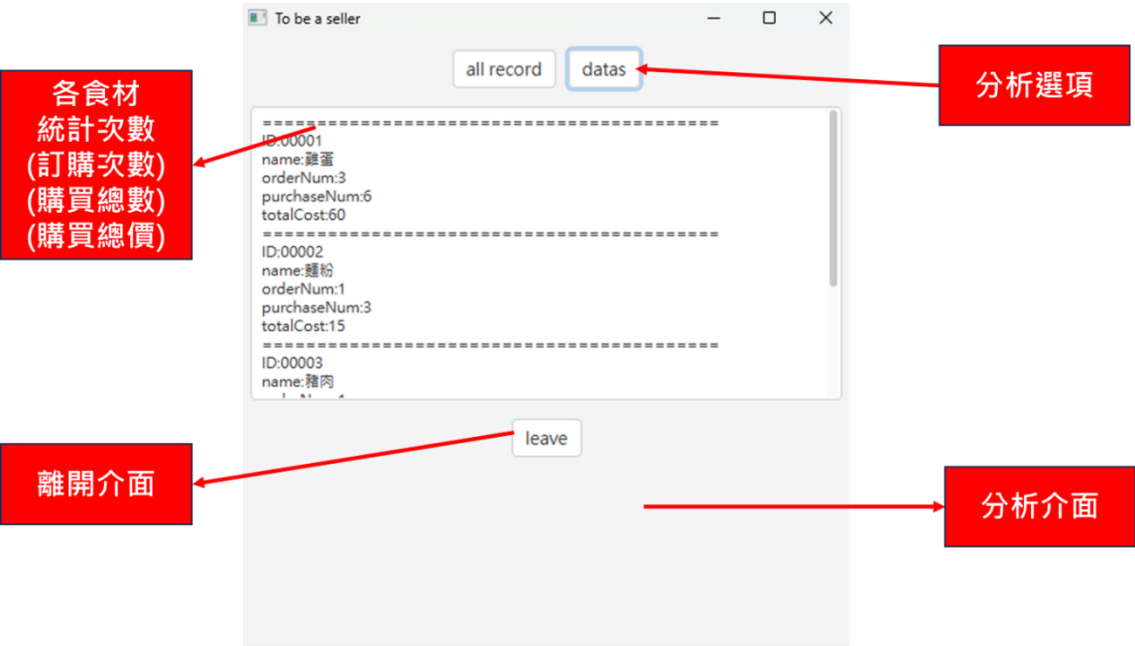


圖 9 統計數據
(資料來源：本人繪製)

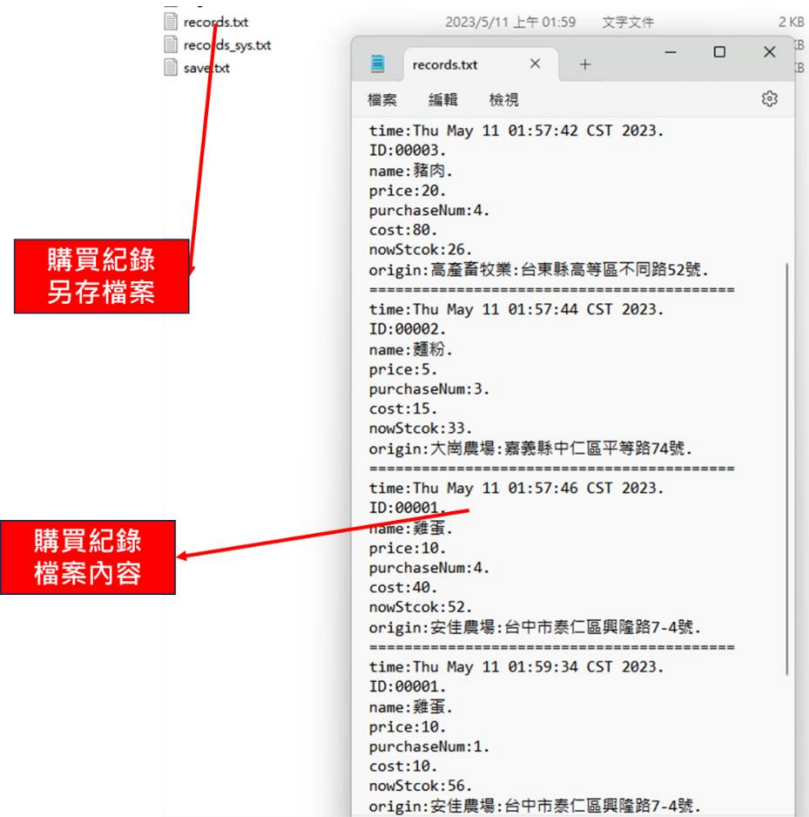


圖 10 購買紀錄檔案
(資料來源：本人繪製)

肆、延伸

由於時間問題，目前僅能完成商城及購買紀錄存檔等……後續功能將持續製作。預計製作功能包含：

一、顧客系統

隨機生成 1-6 人顧客，包含各自點餐內容(各自的表達方式，非正常訂單)、後續加點、心情忍耐度、小費額度等……

二、帶位訂位系統

根據現場空位，讓使用者可點擊帶位，抑或是讓系統自動產生最適排位或所需等待時間。

三、收銀機系統

觀看顧客的表達，進行收銀機點擊商品點餐，並產生發票和銷售紀錄。

四、銷售分析

透過過去銷售資料，進行進一步分析與預測，衡量產品價值。

五、隨機事件

透過隨機事件，造成顧客量不同、食材價格不同、商品可調整價格、特別顧客等……

六、銷售策略

可選擇策略，如：大胃王活動、節日活動、多買特價等……可讓顧客、購買數量等數值上升。

伍、結論

一、差異

在過程中，學習最多的莫過於架構。與以往的作業與小專題相比，多了純程式碼設計畫面的部份，因此整體架構設計除原訂功能外，還得注重畫面呈現與出現的流程與架構。

二、困境

在這段過程中面臨許多困境，主要是在架構上的問題。雖然在設計之前已經設計好架構與流程，但在實際製作時，又經常面臨缺少某個方法，

因此又立刻補充這個方法，這樣子的情況持續多次。設計到後期，有點感覺像是一步一步寫，缺一補一。看似沒有問題，但其實仔細看就會發現，有許多方法是可以一起結合，不必分開寫便可達到同樣效果。

為避免缺一補一的情況持續發生，於是不斷修改架構的設計，但無論如何重新設計，又會在實際製作時遇到相同問題，也只能再次少一補一。因此，這個困境使我在整體設計過程中，不斷卡住停下，再次構思整體架構。除了浪費許多時間外，也容易越設計越感迷茫。

再來，時間問題逼緊了我，在設計過程中，經常會因 bug 而需要停下重新 debug。但在有限的時間下，又因 debug 花更多時間，導致經常需要製作到深更半夜、凌晨 3、4 點才能做到一定進度。

同時，版面設計也困擾許多，色彩設計、版面配置等等……這些在純程式碼設計時，變得相當困難，需要花諸多時間查詢方法，又需不斷測試這個方法是否能達成效果。

總結來說，我認為上述問題是可以透過時間來解決的，畢竟目前都沒遇到過任何無法解決的問題。此外，除了時間問題外，也攸關自身經驗與架構能力。