

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

2016/6/16

CS241001 Software Studio

Final Project Report

Project Name: Zoom and Boom

Team: AC

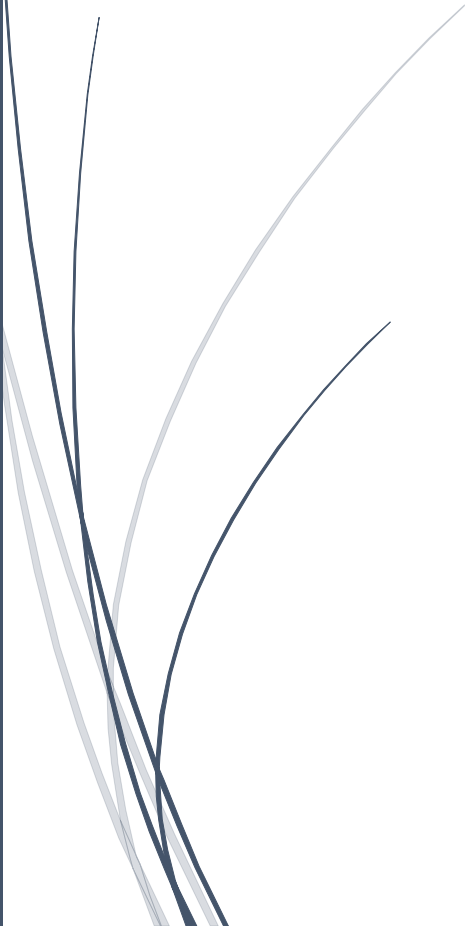
103062310 賴思頻

103062227 張婷雲

103062131 陳映竹

103062181 艾怡華

103062216 范祐恩

Several thin, curved, light blue lines that sweep upwards from the bottom left corner of the page.

壹●概述 Introduction

一、背景與目的 Background and Purpose

(圖一)

一張照片中往往包含太多不必要、或者觀看者不在意的資訊，例如圖一：包含了觀眾、檔板、廣告，然而看照片的人真正關心的應該是被框出的球員。



(圖二)

透過電腦計算分析照片的亮度和色度加以調整後，可以改善這個問題，卻未必總能精準，如圖二：人眼能輕易判斷出這張照片的主角是右圖的車子，然而電腦卻未必。由於車子的顏色與背景相近，導致電腦分析上的困難，若要提升精準度，勢必要耗費大量的運算及時間。



像這類的照片，易造成影像辨識，以及各類影像處理方面的困難，故我們希望能設計一個遊戲，以 Gamification 的理念，誘使玩家幫我們框定出照片中的主角，以期在去除大部分雜訊、縮小分析範圍的情況下，能夠更針對、更精準地處理一張照片。

二、設計概念 Design Concept

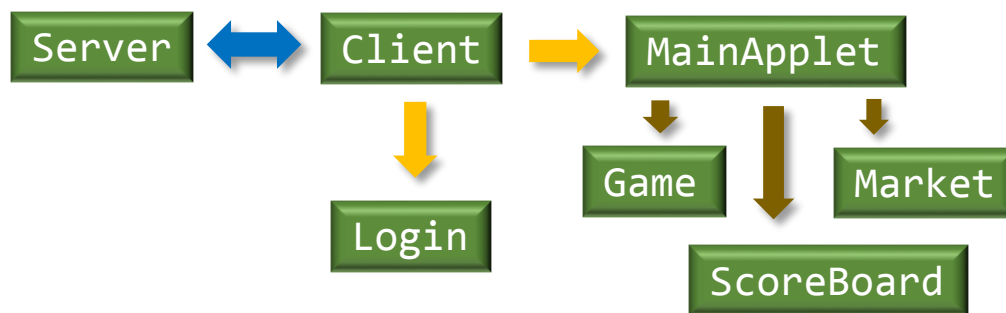
In order to achieve high rates of willing participation from humans, we implement a fun competitive/social game. The player will be shown an image, he/she is required to frame the object in the picture that he/she believes to be the focus of the picture before in 5 seconds to earn money. They can buy the color at market to attack others. Market also provide RANDOM function and shield to defense. The scoreboard will display the online players according to their money in real-time. The state of the player himself/herself is displayed at bottom.

貳●技術細節 Detailed Technique

一、系統架構設計 System Architecture Design

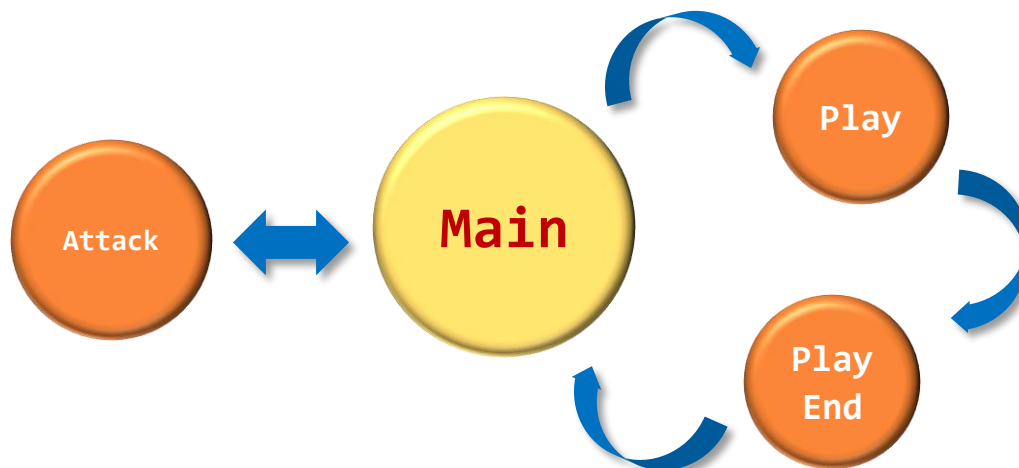
1. 架構圖

Server 與 **Client** 用網路連接；**Client** 包含 2 個 **PApplet**：**Login** 畫面與 **MainApplet**。其中 **MainApplet** 又用三個 **class** **Game**、**Market**、**ScoreBoard** 完成畫面的切割。



2. 遊戲流程

一開始為起始畫面 **Main**。透過點擊開始/暫停鍵進入主遊戲的流程 **Play**，再次點擊開始/暫停鍵結束遊戲，進入 **PlayEnd** 的確認金錢階段，確認完畢回到主畫面。點擊商店區任一品項的按鈕可進入攻擊階段 **Attack**，待完整攻擊流程跑完後才會回到主畫面。於每個階段，不可操作區域將會有半透明黑色遮罩遮蔽，即使點擊也沒有效果。



3. Gamification 核心 – output image file

實作於 **CreateOutput** 物件中，繼承 **PApplet** 但不用另開視窗，只是要藉由 **processing** 的 **draw()** 功能完成圖片截取。

於 **Server** 關閉時呼叫，引進完成之 **answer** (於 **Server** 中存有所有答案的 **HashMap**)。循序叫出 **HashMap** 中的圖檔名稱，將圖片 **load** 進來，並取出對應的 **Answer** 資訊，開一個適當大小的新 **PImage**，用 **pixel map** 的方式將所記錄的資訊擷取下來，最

後使用 **PImage** 內建的 **save()** 將新創圖片存檔。最後會在 `resource/pic_out` 中找到所有有解答資料的 `output` 檔案。



二、建立 I/O 資料庫 Establish of I/O Database

1. Login Window

The Login window was implemented with a **Login** Class extending the **PApplet** Class. Its constructor has a **Client** object argument. An **enum** type variable stores the state of the login applet. 11 states are possible, among them there' s the waiting state that is displayed whenever a query is run on the database. There' s the state for registering, logging, and various different stages for possible failures and successes at login and registering. The **draw()** function inherited from the **PApplet** class displays different components on screen according to the state.

The **Login** class uses the Classes **Textbox**, **SecretTextbox** and **SplashButton** in the object/tool directory for the textboxes and button components, which are initialized in the setup function. The textboxes wrap text and support dots, hyphens, underscores, alphanumeric characters, delete, backspace, tab and enter keys. This is accomplished through a **keyPressed()** overridden function in the Login that calls different methods from the textbox class on the textboxes on screen. The **SecretTextbox** inherits from the **TextBox** Class.

2. SQL Database – players and answers

The queries are made by instantiating a **Database** object. It contains all data to access two MySQL databases, one for the player data and the other for the data on the correct answers of each picture. The databases were created, controlled and checked using the shell of a computer with MySQL installed. MySQL's driver for

queries is included as `mysql-connector-java-5.1.39-bin.jar` in the library of the project. The queries are made to a free server online server. The data on the domain name of the site, the port for the query connection, the username and password of the MySQL account are all hardcoded as private static final strings.

The values stored in the player database are **username**, **password**, **color** (which is assigned randomly to each player on registration), **score**, number of **completed** frames, and number of **"shields"** in inventory. The username value is UNIQUE, and there can't exist two players by the same username.

The values stored in the Picture database are the **name** of the .png file for the picture, the **x**, **y**, **width** and **height** of the focus object and a **"datanum"** variable that keeps track of the times the picture has obtained a player answer for that picture.

The database class provides 5 methods. All 5 functions are implemented creating and starting a new thread that handles the query. `SQLException` and `ClassNotFoundException` (in the event the MySQL driver doesn't work) are handled. They all instantiate a `Connection` object that is always closed before exiting the method (called in a `finally{...}` block). This `Connection` object is instantiated using the `getConnection()` static method from the `java.sql.DriverManager` library and the username, password and host of the MySQL tables. A query is then run using the method `executeQuery()` of an `java.sql.Statement()` object and is stored in a `java.sql.ResultSet` object. We then iterate through the results of the query. The `executeUpdate()` method on the other side updates values inside the SQL table.

FIG 1. User Database

```
mysql> SELECT * FROM player_table;
```

	username	password	score	completed	shield	color
007	boad		0	0	0	2
111	111		0	0	0	2
1111	1111		10	1	0	3
123	123		76	170	0	1
153	153		0	0	0	3
456	456		160	16	0	5
ana	ana		0	0	0	1
adala	adala		0	0	0	3
Anas	Elna		0	0	0	4
hage	hany		0	0	0	3
carlota	lagrade		0	0	0	2
Cindy	1137		660	126	2	6
donald	duck		0	6	0	3
Eva	Eva		1195	55	0	4
Jocelya	Jocelya		415	100	0	2
juan	paloten		0	0	0	4
ladybug	chataoir		0	0	0	2
mickey	mouse		0	3	0	2
minnie	mouse		0	1	0	1
Nathan	nathan		195	40	0	4
Nathan16	nathan		0	0	0	4
papa	perez		20	26	2	3
Tera	Tera		1110	102	0	3
teot	teott		0	0	0	7

24 rows in set (0.30 sec)

FIG 2. Picture Database

```
mysql> SELECT * FROM answer_table;
```

	picture	width	height	x	y	datanum
M0.png		480	421	333	223	0
M1.png		451	439.5	532.5	227.75	0
M10.png		163	132	362.5	206	0
M11.png		455	329	405.5	220.5	0
M12.png		584	350	355	242	0
M13.png		400	346	533	227	0
M14.png		167	158	407.5	223	0
M15.png		307	337.25	285	237.625	0
M16.png		560	440.5	340	227.25	0
M17.png		2.5	5	413.25	82.25	0
M18.png		27.5	14.5	386.125	323.062	0
M19.png		361	327	590.5	203.5	0
M2.png		436	389	471	252.5	0
M20.png		241	334.5	583.5	280.25	0
M21.png		164	146	434	142	0
M22.png		476	436.5	519	229.25	0
M23.png		474	301	475	214.5	0
M24.png		0	0	0	0	0
M25.png		426	432.5	390	231.25	0
M26.png		396	358.5	202	288.25	0
M27.png		56.5	60	449.25	191.5	0
M28.png		477	372	497.5	202	0
M29.png		408	378	363	207	0
M3.png		464	431.5	536	231.75	0
M30.png		229	231	588.5	258.5	0
M31.png		0	0	0	0	0

The method `LoadUserDatabase()` runs a query with the user input in the Login Applet(`username`), checks if the password matches the username and if so creates a new `Player` with values from the database and passes it to the `Client`. If not, the internal state is switched and an error message is shown with a back button to retry login.

The method `newUser()` adds an entry to the player database with the username and password provided by player in the Login Applet in register mode (after double checking password) and initializes the remaining data fields to 0 and assigns a random color, stored in the form of an integer from 1-8. If this elicits an exception (if the player already exists, since the username is a UNIQUE field in the database), the Login Applet displays an appropriate "Duplicated user" message.

The method `updateUserDatabase()` updates the player data when he/she closes the game by updating to the new values after the game session.

The method `LoadAnswerDatabase()` is run every time the server is launched. It reads all the values in the Picture Database and stores them into the Server's public `HashTable<String, Answer>` hashtable using its `put()` method. The string of the hashtable represents the name of the .png picture. The Answer of the Hashtable corresponds to a class inside the object/server directory. It contains the x, y coordinates of the selected area as well as its width and height and the datanum variable.

The method `updateAnswerDatabase()` is called using the Servers' answer hashtable as argument upon closing the server. It updates the picture database's data using the data collected from various players and the area they chose as main focus during the game session .

3. Waiting Window

A window with a small animation created using the Processing library is displayed in a `PApplet` when the queries/updates to the database are made. It modifies the `frameCount` to 500 fps using the `PApplet frameRate()` function and uses the `frameCount` variable of the `PApplet` to change the color every 30 frames to a set of predefined colors and to translate the frame using the Processing `translate()` function. Every 2 frames a rotation is performed. We first use `pushMatrix` to push the data of the transformation to the stack and then rotate the frame using the Processing library method `rotate()` . As the argument for the `rotate()` function, we provide the `frameCount` variable modulo 360 as the angle of

rotation of the frame that we convert to radians. We then draw a "rectangle with a width of 2 pts and no stroke that together with the rest of the rectangles drawn at other frames gives the appearance of a wheel. Finally we pop the transformation data from the stack by calling the *popMatrix()* function.

三、網路 Socket 連接 Connection Between Server and Client

1. 網路

支援跨電腦連線，但 Server 需開在連接有線網路的主機上，且 Client 的目標 IP 要設成 Server 的 IP。目前在 github 上的是設成本機版本(127.0.0.1)。

2. 傳送物件

因為有需要傳送整個 **ArrayList** 的關係，所以用傳 **Object** 的方式來做：

```
objOut = new ObjectOutputStream(socket.getOutputStream());  
objIn = new ObjectInputStream(socket.getInputStream());
```

而讀取就用 *readObject()* 寫 *writeObject()*，讀到的型別是 **Object**，可以用 **if** (讀到的 **Object instanceof** 型別) 來判斷是不是某個型別，之後再強制轉換型別就能拿到傳過來的 **ArrayList**。

3. Protocol 流程

首先登入完畢連上 **Server** 之後，**Client** 傳送自己的資料(型別 **player**)給 **Server**，**Server** 會把這名玩家的資料加入 **playerlist** 中，之後玩家在遊玩的時候，框完圖片會傳送 **"frameEnd"** 給 **Server**，**Server** 經由一連串的判断後回傳 **"Correct"** 或 **"Wrong"** 給 **Client**，**Client** 做出相對應的反應(提示玩家作答正確/錯誤，增加分數、完成數)；遊戲中玩家資料有更新(分數增加或買東西消耗分數)時，也會重新傳自己的資料給 **Server**，**Server** 就會更新所有玩家的玩家列表；**Server** 關閉時會對所有 **Client** 傳 **"terminate"**，**Client** 會通知玩家 **Server** 已經被關閉並關閉 **Client**。

玩家攻擊其他玩家時傳送 **"attack"**、被攻擊玩家名、使用的顏色給 **server**，**server** 再對被攻擊玩家傳 **"be attacked"** 和顏色，被攻擊的 **Client** 根據該玩家還有沒有護盾來決定攻擊是否成功，如果攻擊成功就回傳給 **"picName"**、該玩家現在螢幕顯示的圖片編號、玩家螢幕上的 **Splash** 給 **Server**，**Server** 再轉傳給攻擊者(傳 **"showPic"**、圖片編號、**Splash**)，攻擊者就會在螢幕上看到被攻擊者的慘狀！！反之如果攻擊失敗就會傳 **"protected"** 給 **Server**，**Server** 也會傳給攻擊者 **"protected"**，讓他知道他的攻擊失效了

4. 斷線處理

在 **Thread** 讀取對方傳送的東西時，如果對方失去連線就會跑到 **catch** (**IOException e**)去，以此當做對方失去連線的狀況，在此 **Server** 處理玩家斷線情況(從在線玩家名單刪除該玩家)，**Client** 則用 **JOptionPane.showMessageDialog()**告知玩家 **Server** 已不存在及關閉視窗，類似的原理，如果連線時跳到 **catch** 則可能是 **Server** 不存在。

5. window 關閉時會一併關閉 java application

透過 **window.dispose()**、**applet.dispose()**、**stop()**、**System.exit(0)**來達到，以 **JOptionPane.showMessageDialog()**告訴玩家關閉的原因，如果沒用上述 **mothed** 的話，不正常關閉會造成 java application 還在背景運行，嚴重影響電腦效能。

四、主遊戲 Main Game

1. 開場動畫

在遊戲暫停時以圖片連續切換的方式產生。用每跑一次 **draw()**變數就會++的方式，按數字間隔 **load Image**，控制它出現的速度。選用咖啡杯，因為有 java logo 的意涵。

GIF 連結：<https://j.gifs.com/YEm3v0.gif>

2. 開始/暫停按鈕

使用自訂 **class Button** 實作，**inBtn()**判斷滑鼠是否在按鈕上，判定標準為滑鼠與中心點的距離是否小於半徑，以此判定按鈕動畫(**mouseMoved()**)與執行按鈕功能(**mouseClicked()**)。

按鈕有兩種型態，圓形或是圖片，有各自的建構子與 **display()**，圖片型態的按鈕支援多圖動畫，用 **ArrayList** 存所需圖片，依照情形變換按鈕的外觀，一開始為三角形播放鍵，每按一次會切換成另一張圖片(開始/暫停)。

3. 倒數計時器

The timer is implemented using the **DigitalTimer** class inside the objcet/tool directory. Its constructor takes the parent **PApplet**, the game, the coordinates x,y and the number of seconds after which the timer resets. It creates a small clock image on the upper left corner using methods from the processing library and drawing them inside the parent **PApplet** at the x,y coordinates passed to the constructor.

It uses an instance of the class **java.util.Timer** and calls its **scheduleAtFixedRate()** method. The latter takes a Timer task and two ints as arguments. for the timer task, we create a new **TimerTask** object and overload its run

method to decrement an internal counter if the timer is on. Each time the internal counter reaches the 0 value the timer is reset and the image in the game is changed by calling the Game class method `frameEnd()`. Functions to pause it, reset it resume it and get its value allow us to manipulate the timer objects inside the Game file when pausing/resuming the game.

4. 框定圖片小工具

`inGame()` 用與 `Button` 差不多的方法判定滑鼠是否在框定有效區域內，也就是主遊戲視窗，在其他區塊點擊並不會有綠框出現。

點擊後(`mousePressed()`)觸發 `frameStart()` 標示開始框定，存下當下的滑鼠位置當作綠框左上角的起始點。之後以拖曳(`mouseDragged()`)判定是否正在框定的過程中，於此期間，綠框會隨著滑鼠的移動即使改變大小，右下角為滑鼠位置。

綠框以 `rect()` 實作，填入透明色彩，點擊之後拖曳的方向無限制，但若滑鼠於拖曳時超出主遊戲畫面的區域，綠框只會停留在有效區域內，右下角以有效區域的邊界為準。

鬆開滑鼠(`mouseReleased()`)觸發 `frameEnd()` 標示一次框定結束，並傳送答案資訊給 `Server`。包含換圖機制，隨機挑選一張資料夾裡的題目圖片並更換。可避免點擊被當成框定的情形。`timeout` 發生也會觸發 `frameEnd()`，但不會傳送答案資訊給 `Server`。

5. 暫存錢包與答案動畫

送出框定答案後，`Server` 會回傳答案的正確性，依照結果觸發 `answerCorrect()` 或 `answerWrong()`，答案正確 `Player` 的完成張數會及時增加，兩者都會產生一個 `ImageMessage` 物件，只是裝的圖片不同。

自訂 `class ImageMessage` 實作圖片動畫，於畫面中央產生訊息圖片，之後快速往右下角的錢包縮入，一邊移動一邊縮小直至消失。用 `ArrayList` 存取 `ImageMessage`，可支援同時多張圖片動畫在畫面上，以防使用者圈圖速度過快產生的 `bug`。每新產生一條訊息就 `push` 到 `ArrayList` 裡，定期檢查每條訊息是否已經完成任務而消失，再將不會再用到的訊息從 `ArrayList` 中刪除。

使用自訂 `class Wallet` 實作錢包，基本為一張圖與一個表示現有的錢的字串，具備部分 `button` 的功能，可偵測 `ImageMessage` 是否覆蓋於其上，若被覆蓋會變大並加錢，是顯示答對的錢有確實被加入錢包的動畫。最後要暫停遊戲時，錢包會有短暫動畫，以 `move` 作為 `flag` 區分時期：背景變黑，錢包一邊變大一邊飛入到中央，形成一個按鈕讓玩家確認，按下後會加進右下角的玩家資訊並回復到起始畫面。實作方式為 `Button` 與 `ImageMessage` 的合體。

6. 答案判定

實作於 **Server** 內的 **checkAnswer()**，使用自訂 **class Answer** 存取 **data** 與比對，並將比對結果送回玩家。

Answer 有 2 組答案的儲存空間，以 **dataNum** 決定判斷方式，若資料量 ≤ 3 ，直接算對的答案，在第三筆資料進來時同時算出平均值(3 組除掉變異最大者)當作暫時答案，此後的資料都先經過當下正解的判定，若是對的(誤差在 50 pixel 內)則加入平均計算，重新產生一組解答。

7. 顏料潑濺

呼叫 **addSplash()** 潑濺顏料。即是在 **Splash** 的 **ArrayList** 內新 **push** 進一個 **Splash**，再用 **draw()** 一個個顯示，實作方式與 **ImageMessage** 相同，可支援螢幕上多個污漬共存。

Splash 為自定義 **class**，以一張黑色污漬為基礎底圖。在創建時將圖片 **load** 進來後，用 **pixel** 判斷顏色，黑色部分填上需求顏色，製造不同顏色的污漬。其中 **trans** 代表污漬的透明度，從 255 開始不斷隨時間減少，終至消失。創建時可以設定存在時間，預設 10 秒，時間間隔 $t \cdot 1000 / 255 \text{ms}$ 透明度少一(將 10 秒平分給 255 階段的透明度，時間使用 **processing** 內建的 **millis()**)。

五、商店 Market

在框圖片之外，可以到遊戲視窗下半部的商店購買顏料攻擊其他同時在線的玩家。商店提供各種顏色的顏料，比較特別的是隨機的顏料，以及防衛盾。實作這部分，主要是創建購買顏料和防衛盾的按鈕，購買商品之後，右下角玩家資訊的地方就會透過 **calMoney()** 扣錢，當購買商品錢不夠時就會跳出視窗提醒。

1. 一般顏料

點選之後(**mouseClicked()**)會跳出新視窗(**AttackWindow**)讓玩家選擇要攻擊的其他玩家，並透過此呼叫 **attack()** 進行攻擊。使用 **ImageButton** 陣列來統一處理，不過由於按下去之後的後續處理不一樣，所以把隨機和防衛盾的部分個別宣告。

ImageButton 繼承 **Button**，**Button** 的圖片型態但多加了文字與圓圈，然後 **overwrite display()** 的部分。

2. 隨機

跳過選擇的部分，直接隨機指派被攻擊對象以及顏色(也可能攻擊到自己)。使用 **ImageButton**，直接呼叫 **attack()** 進行攻擊。

3. 防衛盾

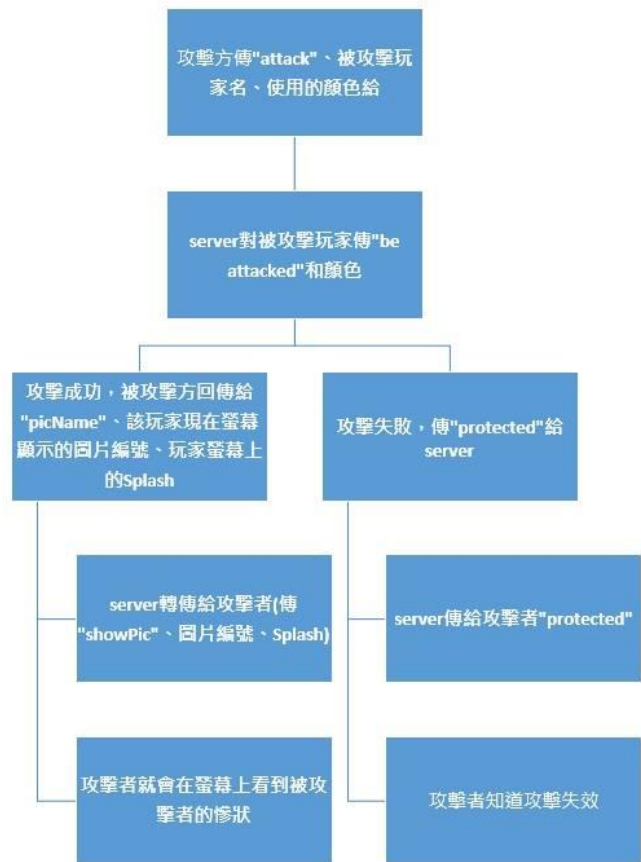
讓自己躲過一次的攻擊。使用 **ShieldBtn**，按下去之後防衛盾會增加，被攻擊時防衛盾就會自動被使用掉。

ShieldBtn 則是由於打算使用不同的圖形，所以另外創建的 **button**，除圖形使用 **quad()** 來繪圖，其餘都和 **ImageButton** 類似。

4. 攻擊視窗與程序

在 **MainApplet** 的 **mouseClicked()** 中創一個 **AttackWindow** (繼承 **PApplet**)，然後把它丟進新創的一個 **JFrame**。按下 **Market** 中一般按鍵會跳出的視窗，使用 **ColorButton** 顯示目前在線的其他玩家，用來選擇攻擊的對象。選定後會呼叫 **attack()**，將攻擊資訊傳給 **Server**，再由 **Server** 轉給被攻擊者，並觸發被攻擊者的 **beAttacked()**。

若玩家備有防衛盾，執行 **useShield()** 並將結果傳回原攻擊者，觸發 **attackNoEffect()** 作為攻擊無效通知；若玩家無防備，執行 **addSplash()** 潑濺顏料並傳回被攻擊者的截圖，實作方式為：回傳自己的圖片名稱 (**String**) 與 **Splash** 的 **ArrayList**，觸發 **showPic()** 的 method：先依傳入的圖片名 (**String**) 將圖片 load 進來，再將傳入的 **ArrayList** 中的 **Splash** 逐一加入並以 class **ConfirmWindow** (extends **PApplet**) 顯示之。



5. 價錢訂定原則(匿名性)

用 **timer.scheduleAtFixedRate(new TimerTask())** 讓他等 100ms 達成延遲讀取，設定好價錢後更新 **Button**，再用一個變數讓他只跑一次就完成設定個別玩家不同的顏色價錢。

6. 水平拉軸

寫於 **HScrollbar**，與下述 **VScrollbar** 類似。

六、排行榜 Score Board

排行榜包含三個部分：顯示玩家個人資訊的區塊、顯示所有線上玩家排行的區塊與代表玩家的圈圈、鉛直拉軸。

1. 玩家資訊

一個玩家的個資定義在 **Player** 的 **class** 裡，為了支援物件的傳送，所以 **implements Serializable**，裏頭的資訊包含：玩家的名字、所屬顏色、完成的圖片張數、總分、擁有的防衛盾數、是否在遊戲中。所有 **field** 都用 **private**，配合 **get/set** 的 **method** 完整封裝。

右下角的區塊會顯示玩家個人的完整資訊，為了美觀，先行使用修圖技術將背景圖的部分區域擦淡，避免干擾到上面顯示的文字。

2. 及時排行榜更新與顯示

右側的排行榜定義於 **Scoreboard** 的 **class** 裡，會動態更新玩家的排名。排序的部分寫於 **Server** 的 **refreshPlayerList()**，裡面包含 **java** 內建的 **Collections.sort()**；並配合 **class CompareScore implements Comparator<Object>** 來比較兩個 **Player** 的 **score**。每當玩家的 **score** 有異動(圈圖賺錢 or 炸人花錢)，便會將更新後的 **score** 傳入 **Server** 端，**Server** 在重新排序完後會以廣播的方式傳送新的排序列給線上的所有玩家。

其中榜上圈圈的顏色即玩家的所屬顏色，圈圈中央有個粉紅色小圈圈的表示玩家自己，讓玩家能一目了然的知道自己在榜上的排行。為了美觀，圈圈半徑會隨著線上玩家人數的增加而線性縮小(有下界)，並且在玩家超過 5 人時會給前三名發黃冠(同分亦發)，皇冠的位置亦經線性計算，能夠準確的戴在圈圈頭上。

每當有一個玩家新加入時，他會把自己的個資傳給 **Server** 端，**Server** 內有一個 **ArrayList<Player> playerlist** 紀錄線上所有玩家的資訊。玩家離線，頭像也會從排行榜上消失。另 **online** 為玩家的在線狀況，若正在進行遊戲，**online** **set**，在排行榜上的頭像是不透明的，反之為半透明，用來標定誰可以攻擊誰不能攻擊。所有關於玩家資訊的更動都會觸發資訊的傳送，將最新資料傳給 **Server** 並廣播更新排行榜。

3. 鉛直拉軸

拉桿的部分寫於 **VScrollbar**，由兩個 **rect()** 組成，並隨滑鼠的狀態 (**mousePressed()**、**over**) 改變 **rect()** 的顏色。**class** 中有 **getspos()** 的 **method**，能隨時取得拉桿當前的位置，使得 **Scoreboard** 上的所有物件能隨拉桿的滑動而移動。

七、其他 Others

1. 文字顯示

🌈 把相同的 text 微調位置以不同顏色顯示，製造一種 text-shadow 的感覺。

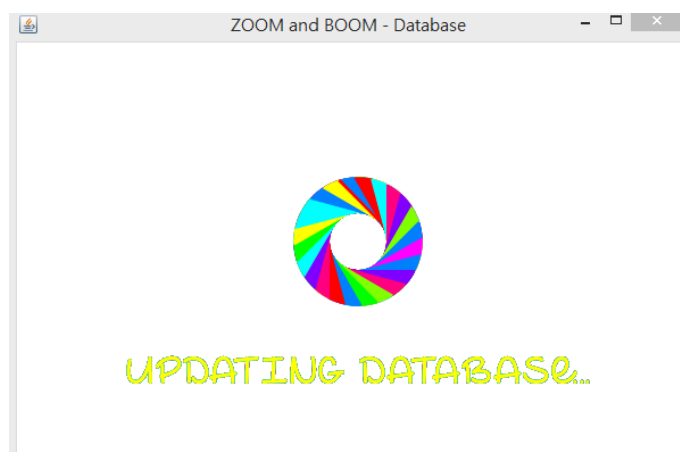
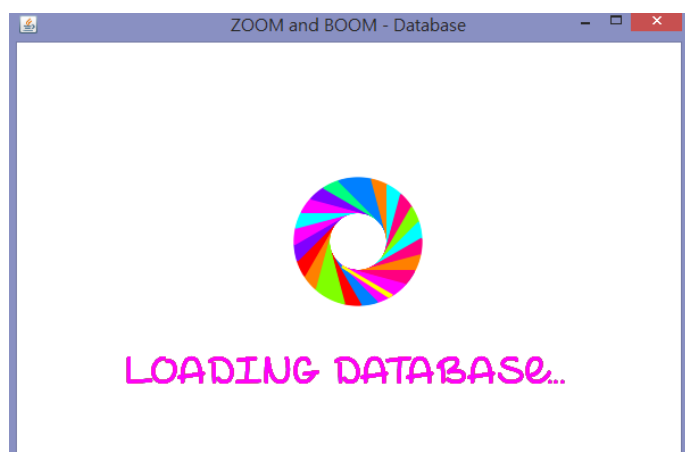
✚ 使用 **PFont** 的 ***createFont()*** 套上許多不同字形。

2. Usability

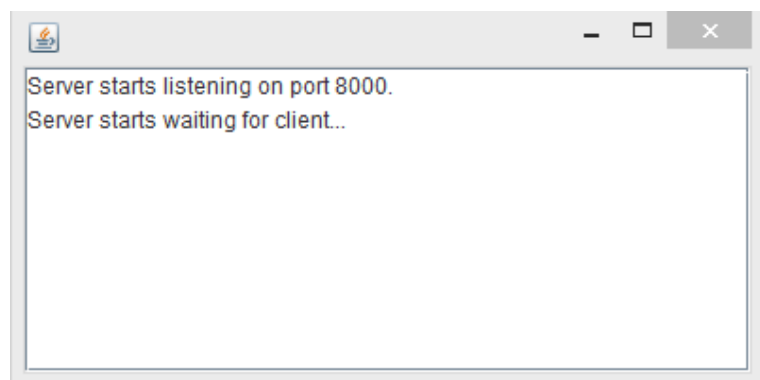
- ✚ We created a small animation window that displays a wheel with changing colors whenever there' s a need for waiting (usually whenever queries or updates to the database are made).
- ✚ The game supports different kinds of errors and provides the error reason to the player. For example, when the Server is not connected, the player is notified about it. Similar errors are prompted to the player when the password doesn' t match the user' s data on the database, when a new user registering is using a name already present in the database, when the registering passwords don' t match, etc.
- ✚ We put effort in developing nice aesthetics to increase the visual appeal of our application. The way the game window is divided into sections help the player to find the information he wants easily and focus only in his area of interest at the moment.
- ✚ The market window was designed in consistency with the scoreboard since the target of attacks are the player themselves (ie. the same background and approximate dimensions are used, as well as the way and the order in which the player names are represented).

參●實作成果

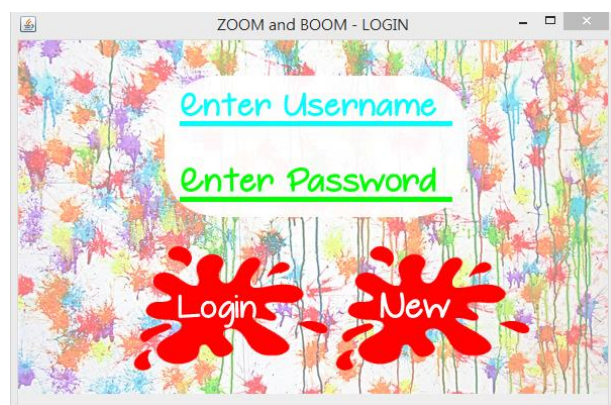
開啟/關閉 Server 的等待畫面



Server



登入/創帳號畫面 – 與各種“意外”



主遊戲畫面



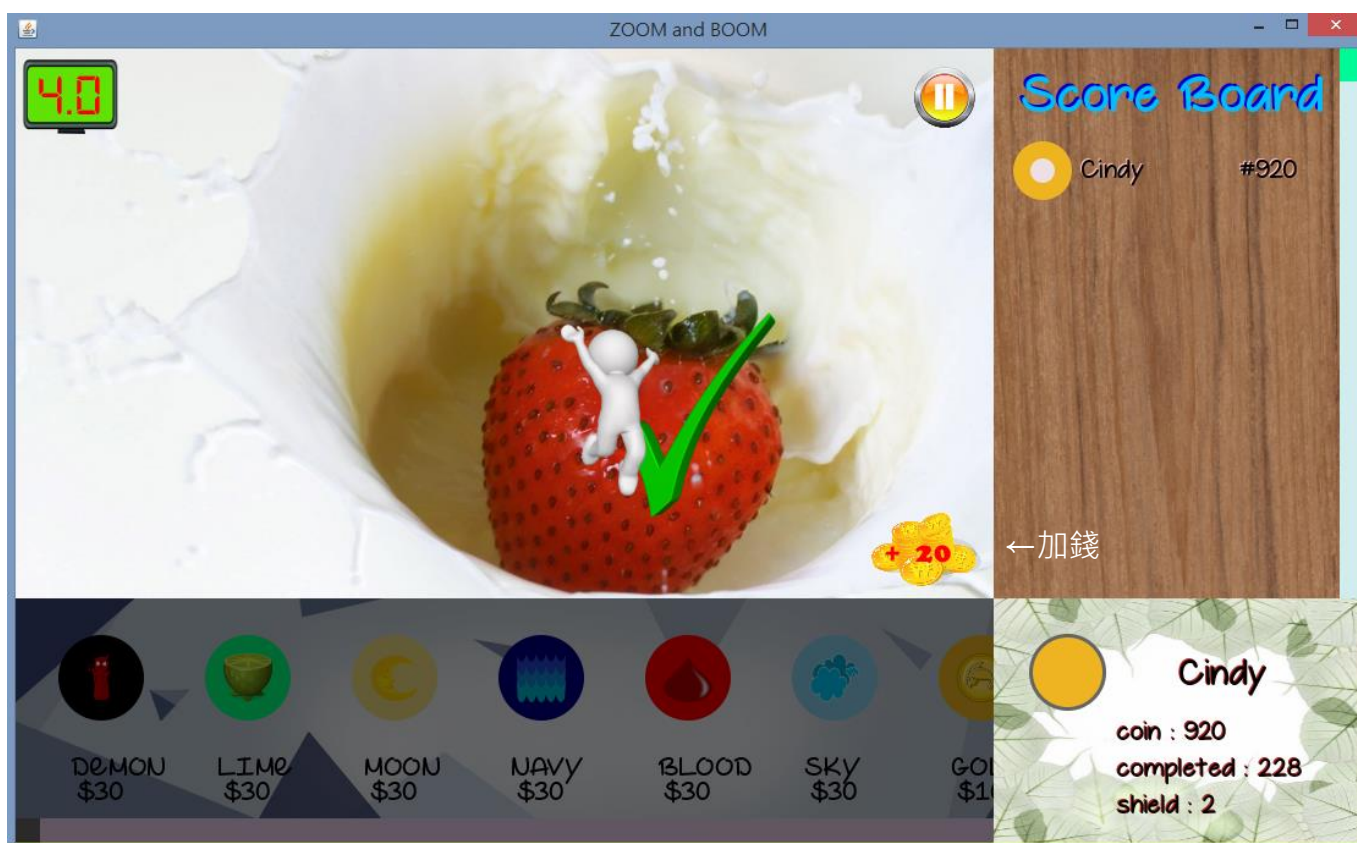
遊戲開始



✚ 框定主角



✚ 框定結果



結束遊戲 – 確認賺的錢



攻擊視窗



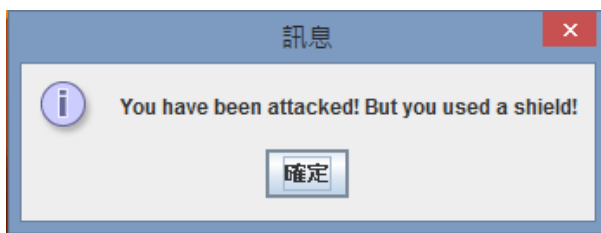
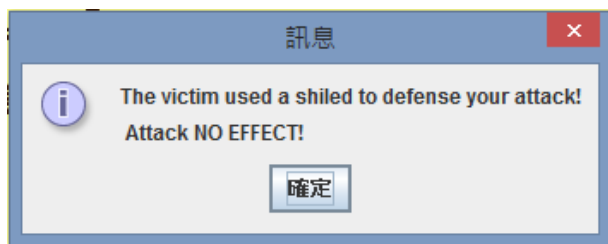
接收攻擊結果 – 對手螢幕截圖



被攻擊畫面



🚩 防禦成功/攻擊無效 – 使用防禦盾的效果



🚩 其他小提醒



一、遇到的困難與內部測試及解決辦法 Problems encountered and solved

Q：在作業中練習到的 **Socket** 只能傳送 **String**，但如果要把自訂 **class** 的所有資訊編碼成一串字串然後再由 **Server** 端解碼非常麻煩。

A：實作傳送物件的功能，改成 **ObjectOutputStream** 和 **ObjectInputStream** 後終於順利的將自訂 **class** 的資訊成功互傳了。

Q：傳送排行榜時，使用自創 **class PlayerList (implements Serializable)**，然而當第二個 **Client** 上線時，雖然第一和第二個 **Client** 玩家都會收到 **Server** 更新完後發送的 **PlayerList**，但兩個 **Client** 收到的 **PlayerList** 不一樣，第一個上線的 **Client** 收到的 **PlayerList** 的長度只有 1。

A：後來發現更新時在 **Server** 端重開另一個 **list**，而非用 **reference** 指向同一個物件，就能解決這個問題了。且由於 **Player** 已 **implements Serializable**，所以可以省略 **PlayerList** 的 **class**，直接用 **ArrayList<Player> list** 傳送整個排行榜。

Q：把 **Splash** 的 **class implements Serializable** 後仍出現 **not Serializable** 的訊息。

A：發現是因為 **Splash** 中含有 **PApplet**、**PImage** 等本身沒有 **Serializable** 的 **class**。在這些 **class** 宣告時加上 **transient** 的關鍵字後成功解決這個問題了。

Q：不知道如何截取螢幕畫面。

A：經上網查詢發現 **processing** 有內建存圖檔相關的 **method**，於了解 **PImage** 的結構後將之與 **save()** 等相關 **method** 結合並應用在螢幕截圖、題目圖檔製造小工具、最後輸出等。

Q：於訂定商品價錢時一直出現 **error** 導致程式無法執行。

A：發現這部份不能放在 **Market** 的 **constructor** 中，因為跑 **constructor** 時還沒登入成功，在取得玩家顏色的時候就會造成 **error**，要放到 **display()** 裡才能讓他成功更新，不過可能是因為取得 **Server** 端記錄的資料需要一點時間，如果一成功登入就讀取玩家顏色也會造成 **error**，用 **Thread** 讓 **Market** **sleep** 的話又會造成 **lag**。

Q：The Login Applet contains a **runFrame()** function that puts the Login Applet inside a **JFrame** and automatically closes when there is a successful login. An empty While loop is implemented to keep the window open as long as there is not a successful login. In some of our laptops the Applet wouldn't close.

A：We fixed this by adding a dummy print statement inside this While loop.

Q：The **DigitalTimer** class implements a timer using the **scheduleAtFixedRate()** method from the **java.util.Timer** Class. Initially, the Game class included a thread that checked the state of the timer, and when the time ran out the **frameEnd()** method would

be called to change the image and reset the time. There was a very noticeable delay between the timer running out and the change of image.

A : This issue was fixed by calling the `frameEnd()` method from inside the `DigitalTimer` method when the 5 seconds for the image display run out.

Q : 主要遇到的困難是在最後把 `code` 合在一起的部分，因為這時就需要了解別人的 `code`。

A : 由於不是同一人寫的，就必須要和原作者討論。

Q : Client 跑起來卡卡的，開工作管理員發現超吃資源...

Q : Some commits were made with computers that were not set up properly for GIT. Editing the Author/Committer name and email was impossible because rebasing the GIT to the problematic commits and editing them would have forced other users to have to reset their local repositories to a commit previous to the earlier changed commit and re-pull all the commits.

二、分工

	賴思頻	張婷雲	陳映竹	艾怡華	范祐恩
Proposal 撰寫					
系統架構設計與整合					
使用者介面設計					
研究可用資源					
內部測試					
使用者評估					
主遊戲程式撰寫					
及時排行榜程式撰寫					
商店機制程式撰寫					
建立輸出入資料庫					
連線機制設計與撰寫					
等待動畫					
蒐集題目圖片					
重製題目圖片小工具					
研究物件傳送並實作					
研究圖片傳送並實作					
遊戲起始動畫					
遊戲狀態分割與防呆					
攻擊匿名性實作(特判玩家顏色價格低)					

隨機功能與防護罩功能實作					
實作答案紀錄演算法並連接主遊戲判斷					
實作攻擊視窗					
整合帳戶系統與主遊戲系統					
連接攻擊機制					
統合排行榜與商店顏色選項					
加入背景音樂					
加入音效					
美化倒數計時器與商品按鈕					
實作拉軸					
研究字型嵌入並實作					
設計文字顯示					
Report 撰寫					

三、參考資料

1. Socket 傳物件
<http://math.hws.edu/javanotes/source/chapter12/netgame/common/>
2. ArrayList 的傳送(reset)
<http://stackoverflow.com/questions/23240064/sending-a-priorityqueue-over-socket-in-java-sends-an-empty-queue>
3. not Serializable
<http://stackoverflow.com/questions/9411292/why-is-my-class-not-serializable>
4. MySQL connector
<http://dev.mysql.com/doc/connector-j/5.1/en/connector-j-reference-configuration-properties.html>
5. Java: Sorting、HashMap、Timer/Task Scheduler
<http://docs.oracle.com/javase/7/docs/api/overview-summary.html>
6. Processing: pushMatrix() /PopMatrix()、Scrollbar、Button
<https://processing.org/>