Socket Programming Project
多人線上即時對戰遊戲-「左居合右狙擊」
開發環境
IDE 使用「Eclipse Java 2018-12」，在同一個 Project「socket」之下，分成「server」與「client」兩個 Package，執行時分開編譯執行。
執行環境為「JRE System Library [JavaSE-11]」，並引用「JavaFX」套件來實作 GUI。
編譯前，要在 Configure Build Path>Libraries>Modulepate 中加入 JavaFX 的 Library。
執行時，要在 Run Configuration 的 VM arguments 中加上這行指令：
--module-path <JavaFX 安裝路徑> --add-modules javafx.controls,javafx.fxml
遊戲構想
「居合斬」，即拔刀術，為一種古代日本武士的極致刀術。在尚未拔刀的情況下，對方無法清楚判斷自己配刀的長度與攻擊範圍，此時將拔刀與斬敵的動作一氣呵成，務求一擊致命，稱為拔刀術。
顧名思義，此遊戲的機制，融合了近距離的劈砍，與槍戰遊戲的遠距離狙擊。
遊戲中大部分動作都有副作用，若先發無法制人，便會置自身於不利的境地，希望藉此營造「敵不動我不動」的「武士道」氛圍。
基本規則
仿效曾瘋行一時的「.io 遊戲」，所有玩家在同一場景中的大亂鬥模式。
雖然所有操作都濃縮在滑鼠移動及左右鍵點擊，但根據情況人物有許多不同的動作。
每個動作有不同攻擊範圍，有範圍攻擊也有單體攻擊，只要被其他玩家擊中一次便直接離場。
操作(客戶端)
收刀/上膛：
將鼠標移到畫面中心，此時人物不會移動，開始收刀/上膛。
如果刀已出鞘且槍也還沒上膛，則鼠標在「人物面對方向」的左方為收刀，右方為上膛，否則自動執行還沒完成的一邊。
移動：
鼠標移出畫面中心部分，人物會自動朝鼠標方向移動。
平砍：
若刀已出鞘，人物移動時點擊滑鼠左鍵，經過極短的延遲及範圍預警後，施展一次中等範圍的劈砍範圍攻擊。
拔刀斬：
若刀已入鞘，人物移動時點擊滑鼠左鍵，經過一段時間的延遲及範圍預警後，施展一次極大範圍的劈砍範圍攻擊，並拔刀。
拔刀砍子彈：
當刀入鞘時，若人物受到「狙擊」，則會立刻自動拔刀並免死。

一瞬拔刀：
當刀入鞘時，若有其他玩家接近至自動拔刀距離(介於平砍範圍的最大距離與拔刀斬範圍的最大距離之間)便會自動拔刀，瞬發斬殺該玩家(單體攻擊)。若兩人皆能拔刀，則互相拔刀招架，無人出場。
狙擊：
槍上膛後，人物移動時點擊滑鼠右鍵，經過一段時間的瞄準及範圍預警後，朝目標方向全地圖狙擊一次，此為命中第一個玩家的單體攻擊。
刺刀：
槍未上膛時，人物移動時點及滑鼠右鍵，瞬發進行一次極短距離的近戰單體攻擊。此攻擊為狙擊槍的刺刀，並不會拔刀。
再來一次：
死亡後點擊視窗，會重新連線創建新角色。
關閉視窗：
直接登出跳 GAME，伺服器端會偵測並將該玩家移除。

程式基本架構

伺服器客戶端互動：
我採用極端中心化的架構，遊戲絕大部分的運算全部都是在伺服器上計算，且伺服器上有一個全地圖的遊戲畫面(上帝視角)。客戶端負責將玩家的操作輸入透過網路傳給伺服器，伺服器經過運算，更新遊戲畫面，然後對每個客戶端，將該名玩家周圍的畫面截圖，然後直接將截圖透過網路回傳給客戶端。最後客戶端直接將截圖貼上在客戶端的視窗上，如此反覆。

Multithreading：
客戶端的部分，由於我是在 JavaFX 的框架下開發，你 Main Thread 最終必須要留給 JavaFX 跑它自己的 loop，且整個 APP 執行過程中都要留給它。
所以，即使客戶端基本上很簡單，沒做甚麼事，我還是得 Multithread。先創建一個 Thread 用來跟伺服器連線及傳輸資料，然後 Main Thread 再 launch JavaFX。
伺服器端的部分就比較麻煩。
學長在 PPT 中介紹的架構，是伺服器有一個 while loop，不斷監聽有沒有客戶端要建立連線。如果有，就產生一個 Thread 專門處理跟這個客戶端的連線，然後 Main Thread 繼續 while loop 監聽有沒有其他客戶端要建立連線。
然而這樣的架構，會遇上跟前面講的客戶端遇到的一模一樣的問題。我的解決辦法也很簡單暴力，就是「雙層 Multithreading」。
跟客戶端一樣，先產生一個「Listen Thread」用來處理「跟所有客戶端的連線」，然後 Main Thread launch JavaFX。接著這個 Listen Thread 再做和學長 PPT 中架構的 Main Thread 一樣的事情，這是第二層 Multithreading。
嚴格來說這只是「邏輯上」有兩層。OS 系統這方面我不敢肯定，但我合理的推測，對 OS 來說，這個 Listen Thread 跟從它裡面再產生的其他 Client Thread 應該是平行的。

物件導向/class：
寫即時遊戲，不用物件導向簡直像在自殺。基本上都要把各個物件寫成一個個 class，這樣邏輯上架構才會明確，不會自己都不知道自己在寫甚麼。
這邊非常值得一提的是我的「player」這個 class。
首先看 JavaFX 的部分。
JavaFX 這個 GUI 套件，它的所有顯示的物件背後都有一個 class。如扇形就有 Arc 這個 class，標籤就有 Label 這個 class 等等。JavaFX 在程式執行過程中要產生新的顯示物件的做法，便是依據這些 class 產生物件，然後將這些物件 append 到一個 list，視窗也有一個代表的 class，而這個 list 便是視窗 class 的成員，代表著視窗的所有子物件。
接下來看 Multitreading 的部分。
依照學長給的示範程式碼，我們會先寫一個 class，這個 class 會「實作」Runnable 這個介面，也就是這個 class 會有一個叫做「run」的方法(method)，然後用這個 class 來創建 Thread。
最後再來看一下邏輯上物件導向的架構。
物件導向的核心思想就是以物件為本。像遊戲中有很多玩家，所以我們就會寫一個叫做「player」的 class。每一個玩家，都用 player class 創建一個物件來代表。Class 定義物件，物件有方法(method)跟成員(data)。像上面提到的所有的操作，例如「拔刀斬」、「狙擊」等等，我們會希望把它寫成 player 這個 class 的 method，然後某個玩家要狙擊時，我們就會說「thisplayer.performSnip()」。又比如玩家有配戴裝備，有一把刀、一把狙擊槍，那我們就會希望把這兩個東西也寫成 class，然後 player class 擁有這兩個 class 作為成員(thisplayer.blade、thisplayer.gun)。
而 class 除了能實作介面，它還能「繼承」其他的 class。綜上所述，當我這樣寫：
public class player extends Circle implements Runnable{ ……
的時候，我的這個 player class，同時兼具了三個角色：

1. 它是在我的程式架構中，在物件導向的邏輯上，直接代表了一個玩家的定義。所有玩家的操作、玩家相關的屬性都在裡面。
2. 它也是在 Multithreading 時用來創建 Thread 的 class，意即在 socket programming 的架構上同時代表著一個客戶端，這個客戶端當然也同時就代表一個玩家。
3. 它還同時是一個 JavaFX 的 Circle(圓形) class。因為 extends(繼承)，所以它也同時直接就是在 GUI 上，用來畫玩家的那個圓形。我的程式也能直接讓 JavaFX 把 player class 當成圓形畫出來。

這是我個人認為這次我的整個程式碼最有看頭的地方，這個做法徹底濃縮精簡了整個架構，而且非常的符合邏輯意義。

程式基本流程

現在是 12/18 晚上 11:55，我來不及畫流程圖了……
所以 Multithreading 的部分我就不講了，不用流程圖的話太複雜了難以表達。

1. 伺服器開機
2. 客戶端執行、請求連線
3. 伺服器接受連線、創建角色
4. 客戶端傳送玩家操作、伺服器計算更新資料與畫面、伺服器回傳畫面(重複)
5. 玩家角色被殺 或 客戶端關閉視窗
6. 客戶端重新請求連線、創建腳色、重新開始 或 中斷連線、伺服器移除該名玩家