FOOP Final Project Report

成員 & 分工

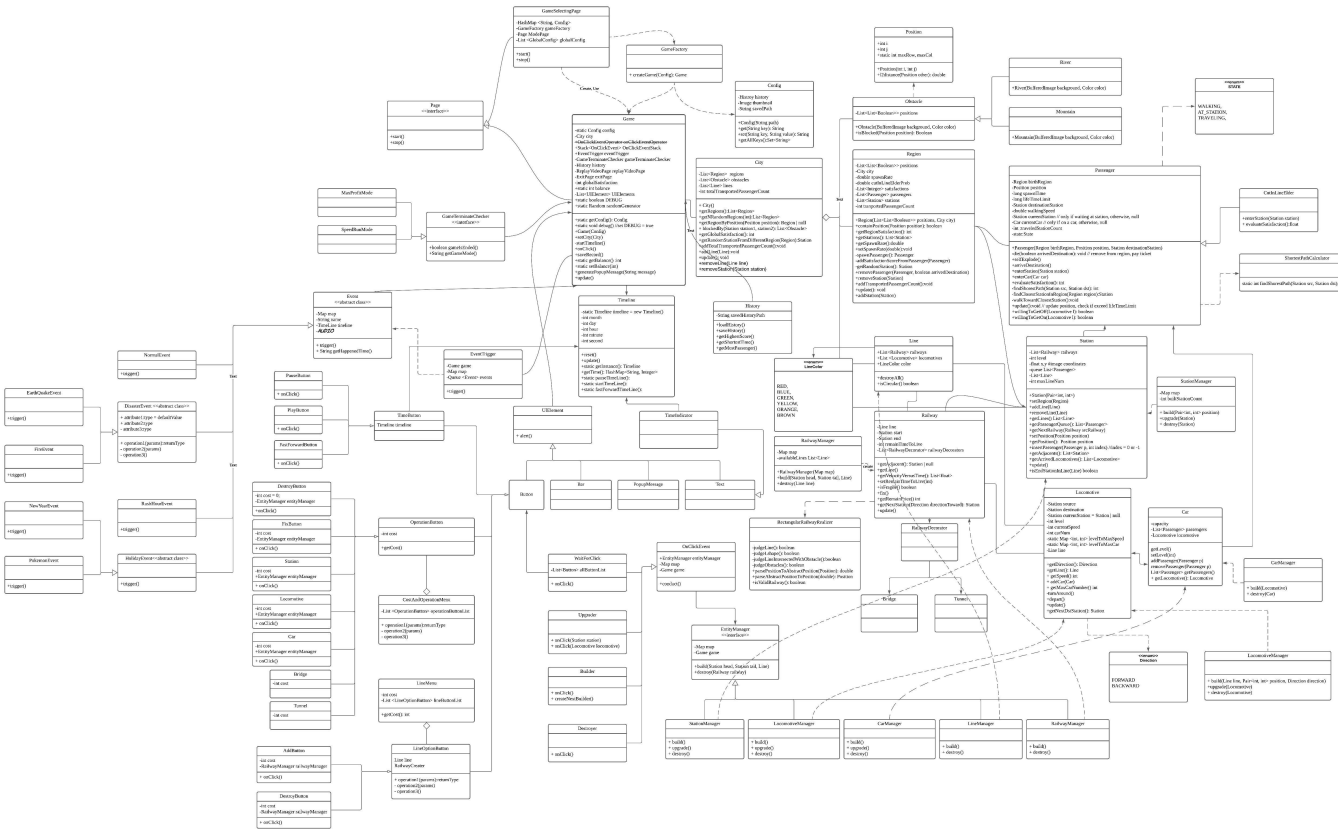
- **劉育嘉**（B06902008）：Event、Timeline、GameTerminateChecker、按鈕相關 UI 介面
- **朱紹瑜**（B06705028）：城市（包含 City、Region、Obstacle、Passenger、Position）、撰寫書面報告
- **黃柏豪**（B06902124）：地鐵系統（包含 Station、Line、Railway、Locomotive、Car、EntityManager）、RectangularRailwayRealizer、ShortestPathCalculator
- **尹聖翔**（B06902103）：OnClickEvent、地鐵系統相關 UI 介面

遊戲簡介

MEowTRO 是一款地鐵規劃遊戲。在城市中，乘客會從各個角落出現，並指定移動至其他區域的目的地車站。遊戲一開始，玩家擁有一筆創業資金，需透過消費投資車站、路線、鐵道、火車頭、車廂、橋樑、隧道等建設資源，建設一套地鐵系統，以將乘客順利送達目的地，收取車票費用。乘客抵達的順利與否，會反映在滿意度上，間接影響每個月的獎金。過程中，亦可能有節慶、過年、火災、地震等突發事件前來攪局，影響各區域生成乘客的速度，或地鐵系統的可靠程度。遊戲模式分為兩種，MaxProfit 模式下，玩家需在有限的時間內賺得最多金錢；SpeedRun 模式則需在最短的時間內達到金額目標。

模型設計

類別圖



重要類別簡介

Game、GameFactory

Game 負責處理遊戲流程，握有城市（City）、遊戲配置參數檔（Config）、事件觸發模組（EventTrigger）、遊戲終止模組（GameTerminateChecker）、帳戶結餘（balance），由 GameFactory 透過 createGame() 方法產生。Game 每次更新（update）會更新時間軸及城市現況。

Config

Config 負責記錄遊戲中的諸多參數，根據指定的 defaultConfig 和 localConfig 配置檔案生成，並提供 get() 方法提供外界透過 key 取得相對應的 value。

Timeline

處理遊戲中的時間軸，以單例模式（Singleton pattern）存在，紀錄遊戲當下的年、月、日、時、分、秒。每一次更新推進一個時間單位（timeUnit），並提供方法取得當下時刻，以及比較兩個時刻的早晚。

City

描述整個城市，記錄所有區域（Region）、障礙物（Obstacle）、地鐵路線（Line）、總共運送的乘客數（transportedPassengerCount），以及所屬的遊戲（game）。提供方法新增及移除地鐵路線、取得隨機一個區域、取得指定區域以外的隨機數個車站等。每一次更新，會更新所有區域以及所有地鐵路線。

Region

描述城市中的一個區域，紀錄所有屬於這個區域的位置點（position）、乘客生成速率（spawnRate）、該區域生成的所有乘客（passengers）、該區域內的所有車站（stations）、區域內乘客滿意度（satisfactions）、由該區域生成且成功抵達目的地的乘客總數（transportedPassengerCount），以及所屬的城市（city）。每一次更新，會更新所有由該區域生成的乘客，以及所有區域內的車站，並且根據當下的乘客生成速率，有機會在區域內的隨機位置生成新的乘客。

Obstacle、Mountain、River

描述城市中的障礙物，分為山（Mountain）及河流（River），該兩個類別均繼承抽象類別 Obstacle，紀錄所有屬於這個區域的位置點（position）。

Passenger、CutInLineElder、ShortestPathCalculator

Passenger 描述乘客，紀錄有生成的區域（birthRegion）、當下的位置（position）、生成的時刻（spawnTime）、壽命（lifeTimeLimit）、目的地車站（destinationStation）、當下所在車站（currentStation）、當下所在車廂（currentCar）、行經車站數（traveledStationCount）、狀態（state）。

乘客依照其狀態，會有不同的行為：在 WALKING 狀態下，每一次更新時乘客以行走速率接近區域內最近的車站，抵達時轉為 AT_STATION 狀態；在 AT_STATION 狀態下，乘客在車站中排隊，若進站的列車行進方向符合其最短路徑，則上車並轉為 TRAVELING 狀態；在 TRAVELING 狀態下，乘客身處車廂內，跟著列車移動，每當列車抵達車站，乘客判斷若繼續搭車不符合其最短路徑，則下車，若抵達目的地車站，則為 ARRIVED 狀態。其中，最短路徑由 ShortestPathCalculator 計算由起始站至終點站最少需行經多少個車站，輔助乘客判斷是否上下車。乘客每一次更新時，會判斷是否接近死亡，則呈現相對應的動畫特效；判斷死亡則從區域中移除。當乘客抵達目的地或死亡時，會將自己的滿意度回報給生成自己的區域。

CutInLineElder 描述一種特殊的乘客，在進入車站時，會插隊排至隊伍的最前端，其餘行為則與一般乘客相同。

Station

描述一個車站，紀錄有所在位置（position）、乘客隊伍（queue）、相連的鐵道（railways）、通過的地鐵路線（lines）、車站級別（level）、通過的地鐵路線上限（maxLineNum）。提供 getAdjacents() 方法，可取得與該車站有鐵道相連的所有相鄰車站。每一次更新會根據當下隊伍中的乘客，重新計算乘客顯示的位置。

Railway、RectangularRailwayRealizer

Railway 描述連結兩個車站的鐵道，紀錄所屬的地鐵路線（Line）、起始車站（start）、終點車站（end）、行駛於其上的所有火車頭（locomotives）、鐵道剩餘壽命（remainTimeToLive）。其圖形轉折點由 RectangularRailwayRealizer 實作，以考慮城市中所有車站，避免穿越。Railway 亦提供方法將行駛於上的火車頭及車廂向行駛方向移動，提供移動後的虛擬位置。

RailwayDecorator、Bridge、Tunnel

RailwayDecorator 描述鐵道的裝飾物，是一個抽象類別。Bridge 及 Tunnel 均繼承 RailwayDecorator，分別描述通過河流及山區所需的橋樑及隧道。

Line

描述由頭尾相連的鐵道所組成的地鐵路線，紀錄有組成路線的鐵道（railways）、通過的車站（stations）、行駛於其上的火車頭（locomotives）、路線的顏色（color），以及其所屬城市（City）。更新時，會更新所有行駛於其上的火車頭。

Locomotive、Car

一輛列車由一個火車頭（Locomotive）連接零至多個車廂（Car）所組成。Locomotive 紀錄有其後連接的所有車廂（cars）、當下所屬的鐵道（railway）、當下的位置（position）、移動方向（direction）、火車頭級別（level）、級別與最大速率的對應（levelToMaxSpeed）、級別與車廂數上限的對應（levelToMaxCar）、狀態（state）、欲上車的乘客列表（getUpQueue）、欲下車的乘客列表（getDownQueue）等。Car 則紀錄有乘坐於內的所有乘客（passengers）、乘客數上限（capacity）、對應的火車頭（locomotive）、當下的位置（position）。

更新時，兩者會根據火車頭當下的狀況有相對應的操作。在 MOVING 狀態下，會透過當下所屬的鐵道取得新的位置，並判斷是否轉向，若抵達下一個車站，則轉為 ARRIVE_DROP 狀態，並逐一詢問其後車廂內的所有乘客是否 willingToGetOff(locomotive)，建立 getDownQueue，並逐一詢問車站內隊伍中的乘客 willingToGetOn(locomotive)，建立 getUpQueue。在 ARRIVE_DROP 狀態下，則從 getDownQueue 中將一位乘客移出車廂，進入車站，若 getDownQueue 空了，則進入 ARRIVE_GETON 狀態。在 ARRIVE_GETON 狀態下，若仍有至少一個車廂有空位，則會從 getUpQueue 中將一位乘客從車站隊伍中移入車廂，若 getUpQueue 空了，則進入 MOVING 狀態。

EntityManager、StationManager、RailwayManager、LocomotiveManager、CarManager

EntityManager 是一個抽象類別，由 StationManager、RailwayManager、LocomotiveManager、CarManager 所繼承，分別負責檢查並執行符合條件的車站、鐵道、火車頭、車廂建造或移除。

Event、HolidayEvent、RushHourEvent、NewYearEvent、DisasterEvent、FireEvent、EarthQuakeEvent、EventTrigger

Event 是一個抽象類別，描述突發事件，紀錄有所屬的城市（city）及發生的時刻（happenedTimeString）。HolidayEvent、DisasterEvent 均為抽象類別，繼承 Event。RushHourEvent 及 NewYearEvent 繼承 HolidayEvent ，紀錄指定的乘客生成速率增加率，觸發時分別影響隨機一個區域及所有區域。FireEvent 及 EarthQuakeEvent 則繼承 DisasterEvent，紀錄壽命殘留比例，觸發時分別影響隨機一個鐵道以及所有鐵道。

EventTrigger 則紀錄有所有的事件，並負責檢查當下時間，在對應時刻觸發事件。

GameTerminateChecker、MaxProfitMode、SpeedRunMode

GameTerminateChecker 是一個抽象類別，負責判斷遊戲終止條件。MaxProfitMode 及 SpeedRunMode 均繼承 GameTerminateChecker。

OnClickEvent、StationBuilder、RailwayBuilder、LocomotiveBuilder、CarBuilder、Upgrader、Destroyer、WaitForClick

OnClickEvent 是一個抽象類別，負責處理畫面中的點擊事件。StationBuilder、RailwayBuilder、LocomotiveBuilder、CarBuilder、Upgrader、Destroyer、WaitForClick 分別處理新增車站、鐵道、火車頭、車廂、拆除物件、等待點擊等事件。

MyButton、StationButton、RailwayButton、LocomotiveButton、CarButton、UpgradeButton、DestroyButton、PlayButton、PauseButton、FastFowardButton

MyButton 是一個抽象類別，描述遊戲中的按鈕，紀錄，並定義有 onClick() 方法。StationButton、RailwayButton、LocomotiveButton、CarButton、UpgradeButton、DestroyButton 均繼承 MyButton，分別描述新增車站、鐵道、火車頭、車廂的按鈕，以及升級按鈕、拆除按鈕，在 onClick() 中會觸發相對應的 OnClickEvent。

PlayButton、PauseButton、FastFowardButton 亦均繼承 MyButton。在 onClick() 中改變 AnimationTimer 以處理正常速度播放、暫停、快進等時間效果。

討論

小組內列下詳細的遊戲規則後，我們發現許多規則的運作模式相似，僅檢查條件不同，或是影響的物件等些微相異，舉例如下：

1. 遊戲終止條件分為「限時最大化利潤」，以及「目標利潤最短時間」分別檢查時間以及結餘，並同樣對 Game 做操作。
2. 山和河流，同樣形成障礙物，但需要不同的鐵道裝飾物。
3. 通勤事件、過年事件、火災事件、地震事件等，都是要在指定的時間影響特定物件（區域或鐵道），但影響的數量相異（隨機一個或所有）。
4. 遊戲畫面中的諸多按鈕，都需要在按下後做建造、拆除、升級等操作。
5. 一般乘客及插隊老人的行為大致相似，僅進入車站隊伍的行為相異。

對於以上這些行為，我們抽象出相似的部分，形成 GameTerminateChecker、Obstacle、Event、MyButton（及對應的 OnClickEvent），以及讓 CutInLineElder 繼承 Passenger。這樣的設計對於日後若需新增其他的遊戲模式、其他類別的障礙物、突發事件、按鈕、乘客類別等，僅需擴充該部分即可，不需更動其它程式碼。

為了捕捉地鐵系統中各物件之間的互動，我們讓地鐵系統的各類別僅處理與系統內其他物件互動的邏輯，對於較複雜的演算法，則委派給其他類別來實作，例如 ShortestPathCalculator 處理最短路徑規劃，以及 RectangularRailwayRealizer 處理建造鐵道時的轉折點選擇。

另外，對於遊戲中諸多常數，例如城市的圖檔、乘客行走速度、列車移動最大速度等，或是事件的時刻，我們將它們抽出寫至獨立的 .properties 檔，在 run time 才運用 Config 進行讀取。這樣在未來僅需對 .properties 進行更改，即可產生不同的關卡

環境

本專案以 Java 11 為核心，並將 JavaFX 11 開源框架用於處理圖形介面。

如何操作遊戲

遊戲畫面分為上列、左按鈕列、右按鈕列，以及中間的城市地圖。

上列

上列顯示遊戲時間，其下三個按鈕由左到右分別為正常速度進行、暫停、快進，可改變遊戲進行的速度。在三個模式下均可建造、拆除地鐵系統中可操作的物件。

城市地圖

綠色和藍色區塊分別代表山區及河流，其餘每一個色塊均代表一個區域。大圖示表示車站，小圖示表示乘客，其目的地為與其圖示相同的車站。

左按鈕列

左按鈕列中的每一個按鈕均對應到一個建設或拆除功能，並顯示其對應金額。由上至下，按鈕及對應功能如下：

1. 新增車站：點擊此按鈕，再點擊城市地圖中的任意區域，即可在該位置建造一個新的車站。
2. 新增火車頭：點擊此按鈕，再點擊城市地圖中現有的任一鐵道，即可在該鐵道對應的地鐵路線上新增火車頭。注意此火車頭無法載客，火車頭後能加掛的車廂數亦依其等級有數量限制。
3. 新增車廂：點擊此按鈕，再點擊城市地圖中現有的任一火車頭，即可在該火車頭後新增車廂提供載客空間。
4. 升級：點擊此按鈕，再點擊城市地圖中現有的任一車站或火車頭，可升級。升級後的車站可容納較多乘客等待；升級後的火車頭則有較快的速度及較大的車廂數上限。
5. 維修鐵道：點擊此按鈕，再點擊城市地圖中現有的任一鐵道，即可進行維修，延續其剩餘壽命。
6. 拆除：點擊此按鈕，再點擊任一車站或地鐵路線。若點擊車站，將拆除該車站及經過該車站的所有路線及行駛於其上的火車頭、車廂，同時在車站等待的所有乘客死亡。若點擊地鐵路線，則拆除該地鐵路線及行駛於其上的火車頭、車廂。
7. 新增鐵道：點擊此按鈕，再點擊欲相連的兩個車站，即可在該兩個車站間建立鐵道。
8. 橋樑：僅顯示金額，不可點擊。若鐵道行經河流，將自動購買橋樑。
9. 隧道：僅顯示金額，不可點擊。若鐵道行經山區，將自動購買隧道。

右按鈕列

右按鈕列依顏色分為六個地鐵路線。建造鐵道時，點擊指定的路線顏色，再點擊欲相連的兩個車站，即可在該兩個車站間建立鐵道。