

遊戲介紹

我們將一款經典的卡片桌遊「皇輿爭霸 (Dominion)」做成電腦版，並且可以透過 socket 與其他玩家在線上進行遊戲。皇輿爭霸是一款構造牌組的遊戲，藉由購買卡片擴展牌組，組成不同的combo，獲得分數而贏得勝利。

遊戲中的卡牌可以大致分為三類：

1. 行動卡 (白色)

- 打出行動卡需耗費一點行動值，打出後可以執行牌上的效果
- 攻擊卡-行動卡特例，此卡會有對其他玩家發動的特殊效果
- 應對卡-行動卡特例，當有玩家打出攻擊卡時，其他玩家可以展示手上的應對卡以發動應對卡的特殊效果

2. 錢幣卡 (黃色)

- 每一種錢幣卡有不同的價值，可以透過打出錢幣卡購入新卡牌

3. 分數卡 (綠色)

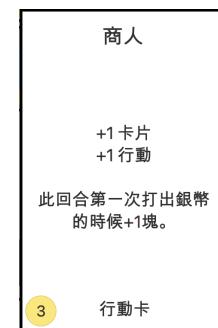
- 每一種分數卡有不同的分數，遊戲結束時分數最高者獲勝
- 特例：詛咒卡 (紫色)，此卡代表的分數為負數



(左下角為購買卡片所需的金額)

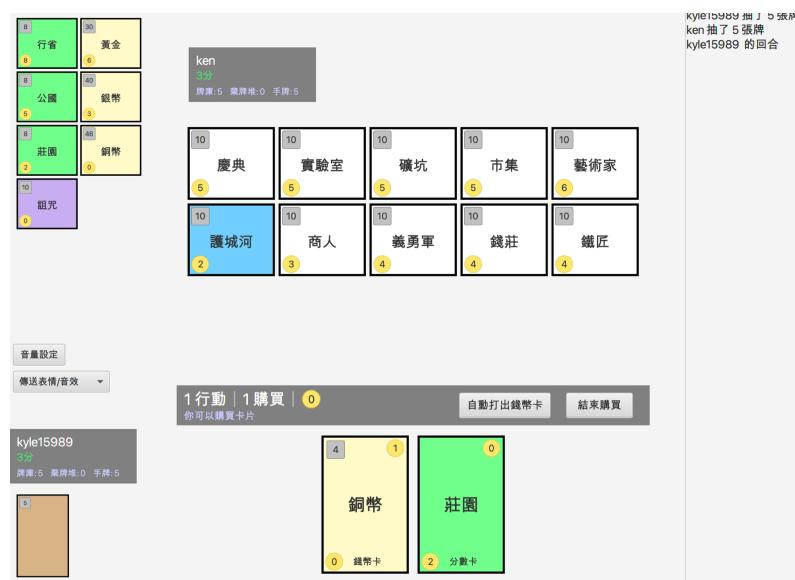


(左上角為卡片的張數)
(右上角為卡片的價值)



(在遊戲中按右鍵能展示
卡牌的完整敘述)

遊戲畫面：



遊戲畫面有三大區域，供應區、手牌區，與棄牌堆/牌庫區。

初始設置：

每個玩家的牌庫初始會有七張銅幣 (價值1塊) 與三張莊園 (價值一分)，並在遊戲開始時抽出五張作為初始手牌，遊戲採回合制運行。

一個回合可以分成以下幾個階段：

1. 打出行動牌

- 玩家可以在此階段打出行動牌
- 玩家每回合初始有一點行動值，打出一張行動牌需要花費一點行動值
(預設只能打出一張行動卡)

2. 購買卡片

- 玩家可以把手中的錢幣卡全數打出，獲得相對應的金額
- 玩家可以用錢購買場上供應區的卡牌，買的卡牌會放入棄牌堆中
- 玩家每回合初始有一點購買值，購買一張卡牌需要花費一點購買值
(預設只能買一張牌)

3. 重置

- 玩家把所有打出的牌丟至棄牌堆 (行動卡、錢幣卡等)
- 玩家把所有沒用的手牌丟至棄牌堆
- 玩家從牌庫重新抽出五張牌，如果牌堆空了，將棄牌堆洗回牌堆
- 換下一位玩家的回合

我們可以發現，不管是打出的牌還是購買的牌都會移至棄牌堆，牌庫沒牌後最後再回到牌庫中。因此你打出的牌不會消失，所擁有的卡牌數量會越來越多。並且每回合初始手牌都是五張，因此購買更高價值的錢幣卡 (銀幣、黃金)，可以提高單位手牌價值，並提升購買力。另外，分數卡只有在遊戲結束時算分才有效果，在手中是沒有任何用處的，因此平衡分數卡的數量也是重要的策略考量。

當供應區的牌的其中三疊被購買完，或是行省 (最左上角，6分的分數卡) 被購買完時，遊戲結束，最多分數的玩家獲勝，一個玩家的分數是他所有擁有的分數卡的分數總和 (不管在牌庫、棄牌堆或手牌中都算)。

如何連線遊玩

同裝置

創建房間:port 輸入任意閒置的 port

加入房間:ip 輸入localhost, port 輸入創建時輸入的 port

同區網

創建房間:port 輸入任意閒置的 port

加入房間:ip 輸入 host 的 ip, port 輸入 host 的 port

不同區網

如果裝置有 public ip, 可以直接輸入 ip 連線。

沒有的話可以使用系上工作站做 port forwarding, 指令如下。

```
ssh -R <任意 port>:localhost:<port> <id>@140.112.30.40
```

```
ssh -g -L <port>:localhost:<任意 port> <id>@140.112.30.40 (在工作站上)
```

上下兩個任意 port 要輸入一樣的 port, 如此就可以用以下方式連線:

創建房間:port 輸入<port>

加入房間:ip 輸入工作站的 ip (140.112.30.40), port 一樣輸

遊戲實作介紹

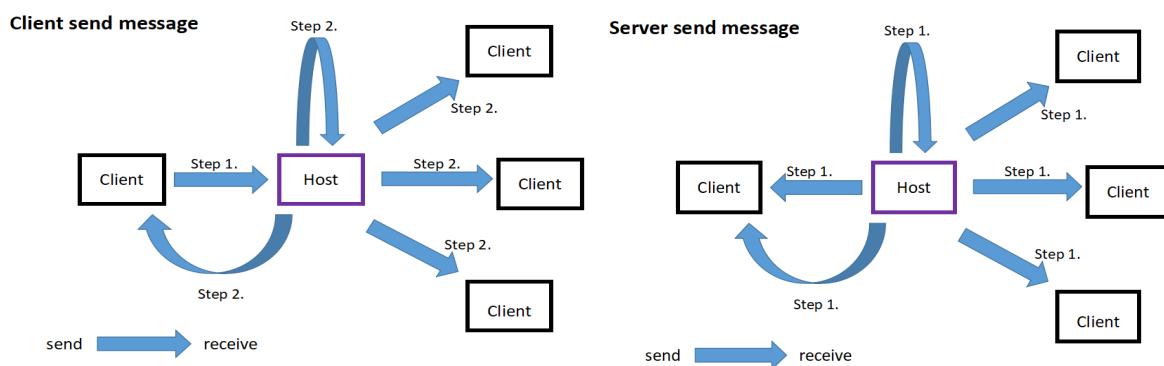
使用的 package:

我們用 javafx 當作 UI 的 library。

多人連線:

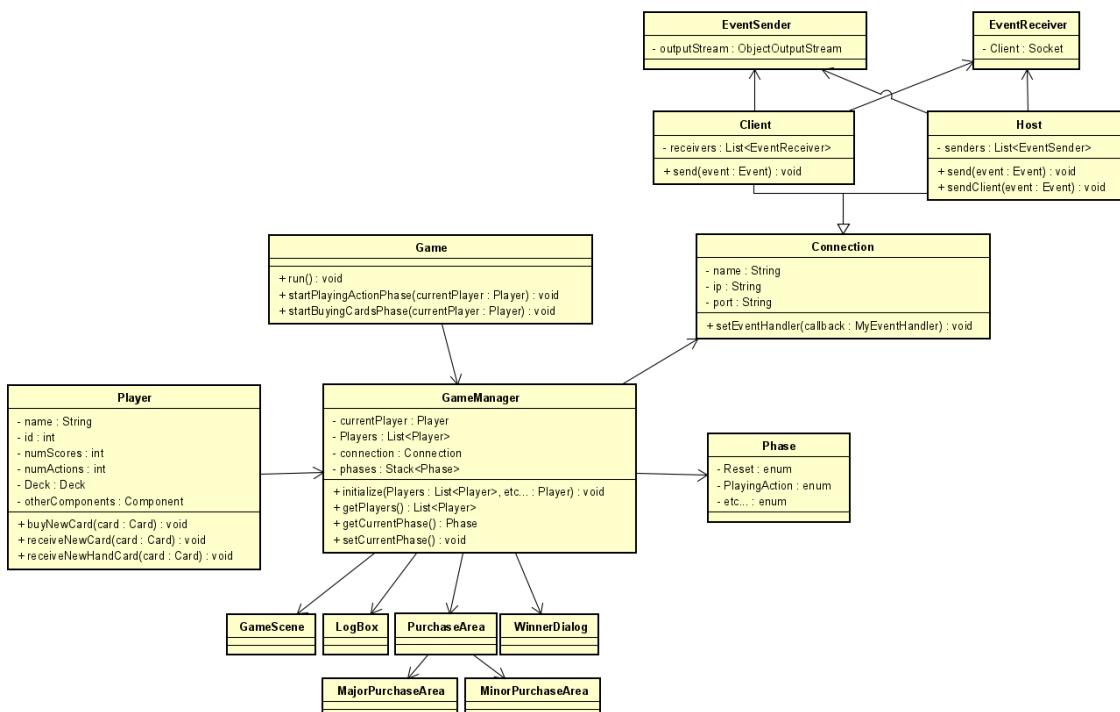
我們使用socket來進行連線，會有一個玩家負責創建房間作為host，其他的玩家為client。每當一個client在遊戲中做出人為的選擇(e.g.出牌)，就會送出一個Event給host，當host收到訊息後就會轉傳給所有client。如果是host做出選擇，則會直接寄出Event給所有client，如此就能在玩家之間進行溝通。

因此，我們建立了一個base class, Connection，並讓Host與Client繼承，在玩家選擇創建或房間時生成對應的class。



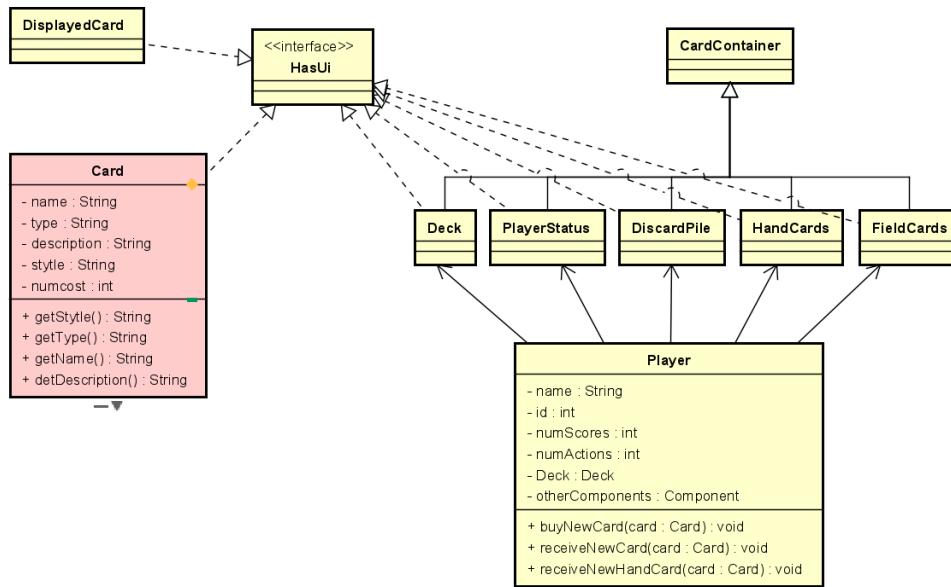
Game 與 GameManager

Game 這個 class 掌管的是遊戲的 game loop，並且會獨立在一個 thread 跑。而 GameManager 則是一個static的class，負責掌管整個遊戲的重要資源，像是所有 Player，現在行動的 Player，或是遊戲正執行的階段等等。在遊戲中，GameManager也會被注入 Connection，因此在遊戲中傳送與接收 Event 都是透過 GameManager。



Player與UI：

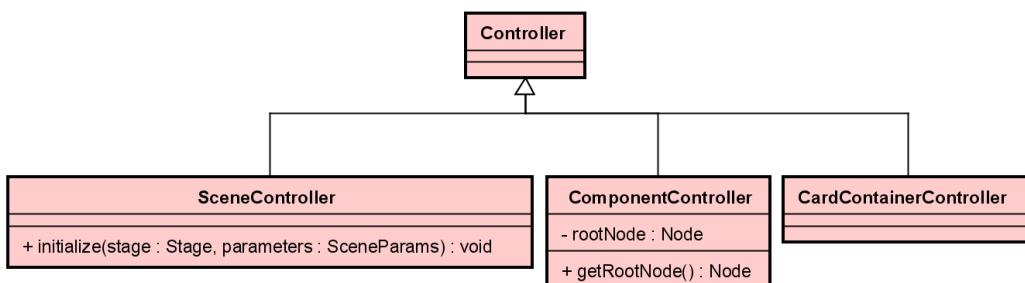
我們使用一個 **Player** 來代表一個玩家，每個玩家都會有各自的 **Deck** (牌堆)、**Discard Pile** (棄牌堆)、**HandCards** (手牌)、**ActionBar** (行動列)、**PlayerStatus** (狀態列) 與一個共用的**FieldCards** (打在場上的牌)，關係大致如下。



實作 Player 的困難點在於，假如現在的遊戲有三個玩家，那麼這個遊戲會以三個不同的process獨立運作，而在每個玩家的視角中，他的遊戲裡只會顯示他自己的手牌與牌庫等等。所以我們必須要將 Player 的 UI 封裝起來，並在遊戲開始前就決定這個 process 中要顯示的是哪一個 Player 的 UI。等到遊戲開始後，無論現在的 Player 有沒有顯示 UI 都可以透過一樣的 method 執行動作。

舉例：有兩個玩家 A 與 B。當 A 抽了一張牌，在 A 的遊戲裡會出現新增手牌至畫面的動畫，而在 B 的遊戲中只能看到 A 數字上的增加，然而他們都是對 **Player** 使用一樣的 **method**，唯一的差別只有在初始化遊戲時分別對不同的 Player 開啟UI而已。

在遊戲中，多數的物件都有類似的問題，有些時候要顯示 UI，有些不用。因此我們把所有有 UI 的物件都對應了一個各自的 **Controller**，負責它 UI 的顯示，將邏輯與UI切分開來。Controller 總共可以分成三類，**SceneController**、**ComponentController** 跟 **CardContainerController**。正如他們的名字，**SceneController** 負責控制 scene 的 UI，**ComponentController** 控制物件的 UI。**CardContainerController** 則是掌管手牌、牌庫等等的 UI，它們比較特殊，本身並沒有 UI 可以展示，負責的是把裡面的Card 放到對的畫面位置並展示動畫。



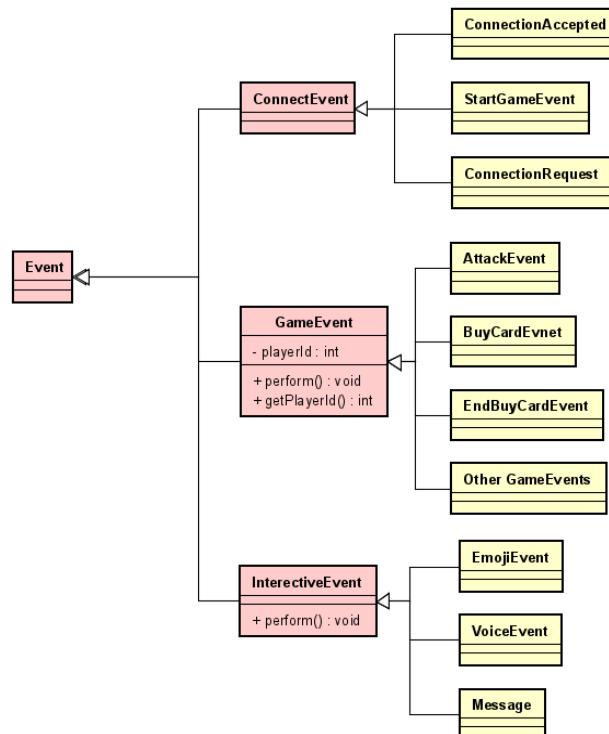
PlayerAction:

遊戲中，每個動作都是要以玩家為主體去實行，如：出牌、抽牌等，若將各種動作零碎的寫在 Player 這個 class 中的話，那程式碼就會凌亂不堪，且若要新增動作，就很可能要改動原來的程式碼。所以我們利用了 Dependency Injection，定義了一個abstract class，叫做 **PlayerAction**，即代表了 Player 動作的抽象概念。每要增加一個動作，就增加一個 Subclass 繼承它。在要執行某個動作時，傳入相對應的 class 到Player 的一個以 PlayerAction 為參數的 method 中。如此一來，Player 就只依賴abstract class 的 method，只要負責執行這些 PlayerAction 的 method，而不用知道他是如何實作的。以這種方法我們還可以很容易擴充新的 Player 動作，非常符合 OCP，且動作的職責與 Player 的職責得以分開，也達到了 SRP。

Event:

我們把每個玩家的人為選擇對應一個Event。可以分成三大類，**GameEvent**、**ConnectionEvent** 與 **InteractiveEvent**。

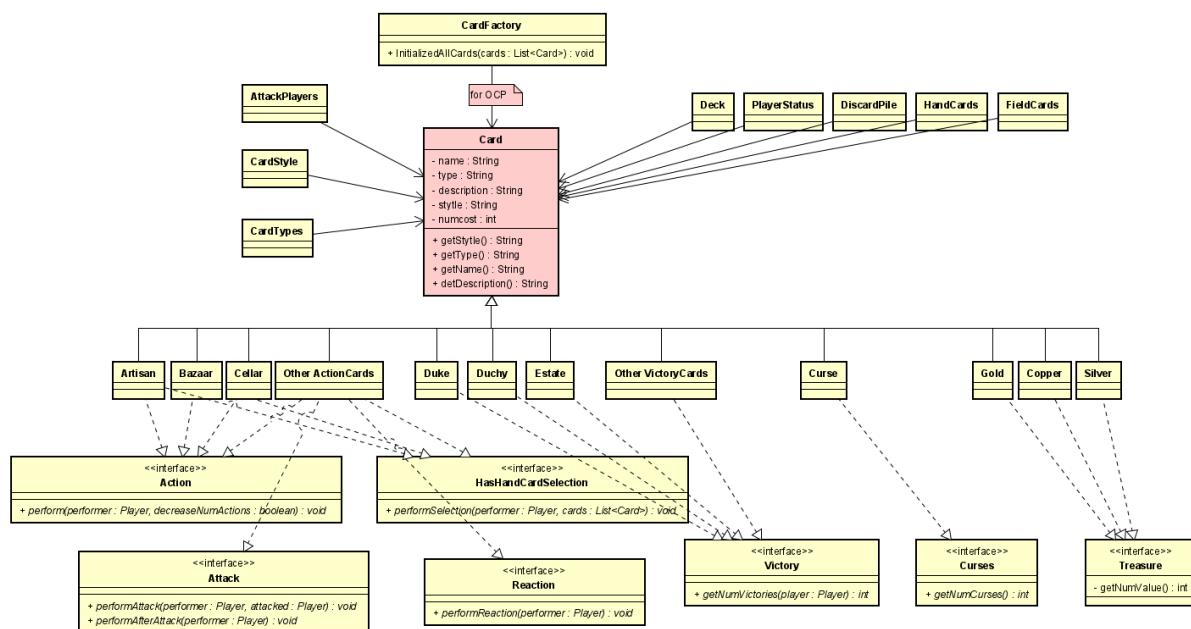
1. **GameEvent** 為遊戲進行中，因應遊戲流程或者是卡牌的效果，由玩家所觸發的事件，作為溝通橋梁的訊息，讓遊戲的流程透過這些event控制。例如當玩家進行到購買階段時，當點選了要購買的卡牌，就會送出一個BuyCardEvent，讓Player 可以做出相應的購買卡片的行為；或者是當玩家在執行完當前購買回合時按下了結束購買之後，就會觸發 EndBuyingPhaseEvent，讓遊戲進行到下一個phase。
2. **ConnectEvent** 則為在遊戲開始之前，要創立房間以及加入房間時，所傳遞的讓不同遊戲端之間可以互相聯繫的event。
3. **InteractiveEvent** 則是在遊戲過程中，可以隨時觸發的事件，如：嘲諷，可以隨時點按讓所有玩家受到嘲諷。



卡牌：

卡牌可以說是這遊戲的核心，我們總共實作了27張卡牌。卡牌的 base class 是 Card，並且有許多不一樣的 interface 去代表不同的種類，像是 Action (行動卡)、Treasure (錢幣卡)、Victory(分數卡)。還有一些附屬的種類，像是 Attack (攻擊卡)、Reaction (應對卡)，或是根據行動效果分出的 HasHandCardsSelection 等。

這裡實作最困難的細節在於每一張牌都有一種到多種的可能的屬性，如一張牌可以同時是行動牌又是應對牌，考量到同一種屬性的牌都能抽象成一種概念，要在 Java 中能做到這點的，就是利用 interface 和 polymorphism 來達成。在每張牌相應 interface 的 method 中實作該 interface 所帶有的概念，如：在所有的攻擊牌中雖然用的都是不同的攻擊方式，但都會有 performAttack 這個 method，代表著攻擊這個抽象。在各個 Interface 的 method 之間要可以相互連結的話，靠的就是上面所述的 event，透過一個 event 的觸發讓 event 相對應的 interface method 可以被 invoke。



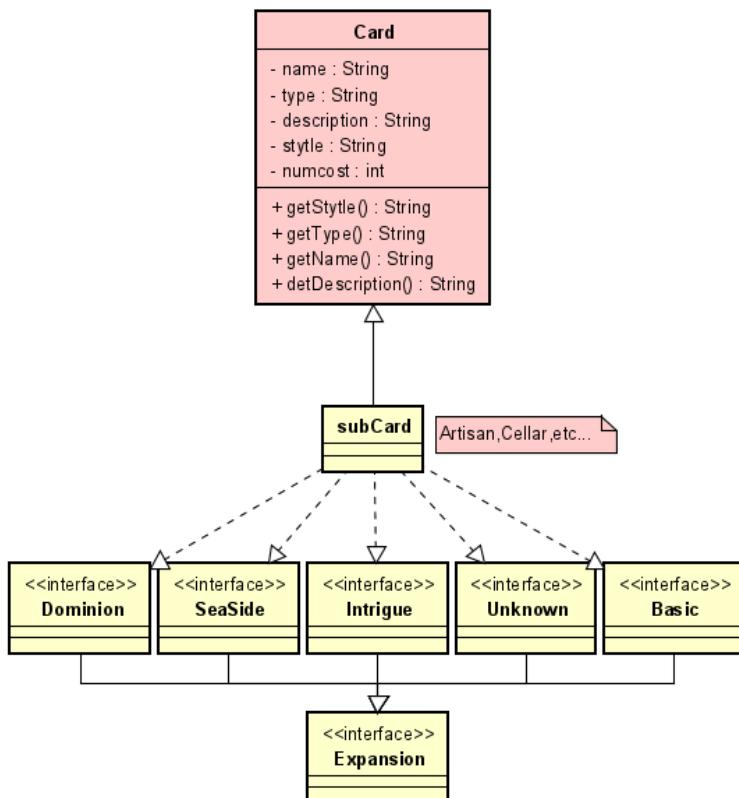
擴充卡牌 (實作OCP):

皇輿爭霸擁有非常多不同的擴充，每種擴充都會有全新的卡片，帶來不同的遊戲體驗。我們總共做了兩組擴充 (除了經典組)，因為時間上的不足因此只有各實作幾張卡片而已。在遊戲開始前，房主可以設定這場遊戲要啟用的卡牌，在那邊會列出目前已實作的全部卡牌。並且選擇完成後，會以注入的方式將選擇的卡片傳送到遊戲中。



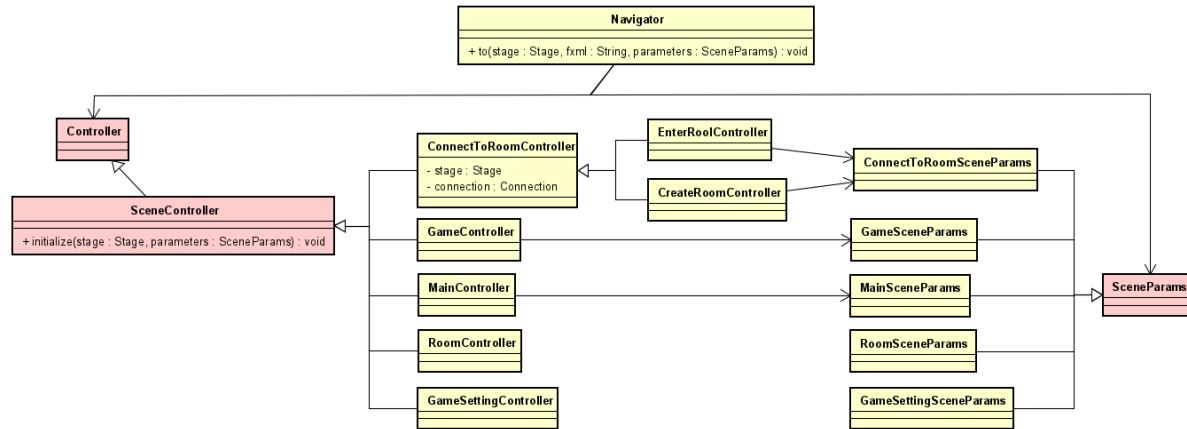
在增加新的擴充與卡牌上，我們做了 OCP 的處理。當實作完新的卡片/擴充後，只需將 class 新增在 Main 的 initialization procedure 裡，在設定頁面中就會出現新的牌/擴充。

實作上，每一種擴充都會有一個 interface，像是 **Dominion** (經典牌)、**Intrigue** (暗潮洶湧)、**SeaSide** (海國圖誌)，並繼承 **Expansion** 這個 interface，而每張卡牌也會 implement 相對應的 Expansion。並且我們有一個 static class **CardFactory**，會在 Main 裡被初始化，注入全部的 Card 與全部的 Expansion，如此設定頁面就能在不耦合的情況下拿到全部的擴充與對應的卡牌。



Scene 之間的切換

使用 javafx 進行 scene 之間的切換需要許多程式碼，因此我們用了 **Navigator** 這個 class 來控制 scene 之間的切換。然而每個 scene 需要傳入初始化的參數都不相同，因此我們使用一個 class **SceneParams**，把每個 scene 需要的參數封裝成一個 class 並繼承 **SceneParam** 來抽象化不同的參數。並且讓 scene 的 controller 都繼承 **SceneController**，提供一個透過 **SceneParams** 來初始化的 method。如此，就能把繁瑣的切換程序簡化。



優點

經過數次重構，我們把各個 class 之間的職責重新分配過，盡量讓 model 能符合 SRP 的原則，讓職責不會分散在程式各處，提高了各個 class 的內聚性，讓程式更好讀懂及更改。在可能有 behavioral variation 的地方，我們依循著 OCP，讓複數行為抽象成概念，讓只要能符合舊有概念的新行為可以不必修改舊有程式碼，讓擴充更加容易。

缺點

因為 model 某些地方的設計的因素，參數需要層層傳入才能抵達真正發生影響的地方，可能在職責分配處還可以再稍微多花心力設計。且因為層層設計的關係，相對的較為複雜，overhead 也比較大。

組員/分工：

朱瑞斌 **B08902003**:

寫程式，寫程式，寫程式，寫報告

蘇浚笙 **B08902042**

設置音效與表情，應用程式中斷處理措施，新增 UI 介面，demo 影片

謝辰陽 **B08902076**

實作卡片，程式重構，寫報告

吳冠磊 **B08902082**

寫報告，畫圖，提供音效，demo 影片