FOOP Final Project: Rich Man

The team members' names and school IDs

- B06902030 邱　譯
- B06902058 吳崇維
- B06902073 張華恩

How to play Rich Man

- 介紹：
  這是一款大富翁遊戲，玩家們透過輪盤決定移動步數，試圖佔領更多的地產並升級房屋，亦可透過各種卡片功能躲避他人地產，掠奪金錢，成為最後的大富翁來獲得勝利。
- 遊玩：
  每一個回合中，玩家可以進行以下的操作。
  - 使用卡片
    - 透過按左右鍵可以選擇卡片，按`ENTER`使用卡片，選完卡片如果還需選擇作用對象，一樣透過左右鍵與`ENTER`操作。
      - 輪盤控制卡：可使輪盤轉速變慢
      - 烏龜卡：可使一名玩家進入烏龜狀態（只能前進一格）三回合
      - 搶劫卡：搶走一名玩家 20% 存款
      - 去狀態卡：可去除一名玩家身上的狀態
      - 加速卡：可使一名玩家進入加速狀態（前進格數三倍）三回合
      - 停留卡：此回合停在原地來代替輪盤
      - 轉向卡：可使一名玩家更改前進方向

        - 瞬移卡：可使一名玩家移動到一個隨機地點
    - 轉輪盤
        - 在你的回合中，除了正在選擇卡片作用的對象時，可以隨時按下`SPACE`來啟動輪盤，啟動後再按一次`SPACE`停止輪盤並根據轉到的數字移動。
    - 觸發 event
        - 地產
          如果地產沒有擁有者，玩家可以選擇是否購買；如果有擁有者，則會根據玩家是不是擁有者來選擇升級或是被罰錢。
        - 野狗
          會跳出玩家被狗追的畫面，然後被移到醫院三個回合。
        - 暴走族
          會跳出玩家被暴走族追的畫面，然後被移到醫院三個回合。
        - 商店
          會跳出商店畫面，裡面有卡片與售價，玩家可以透過左右鍵與`ENTER`選擇購買哪一張卡片。

    event 結束後，玩家的回合結束，輪到下一位玩家。

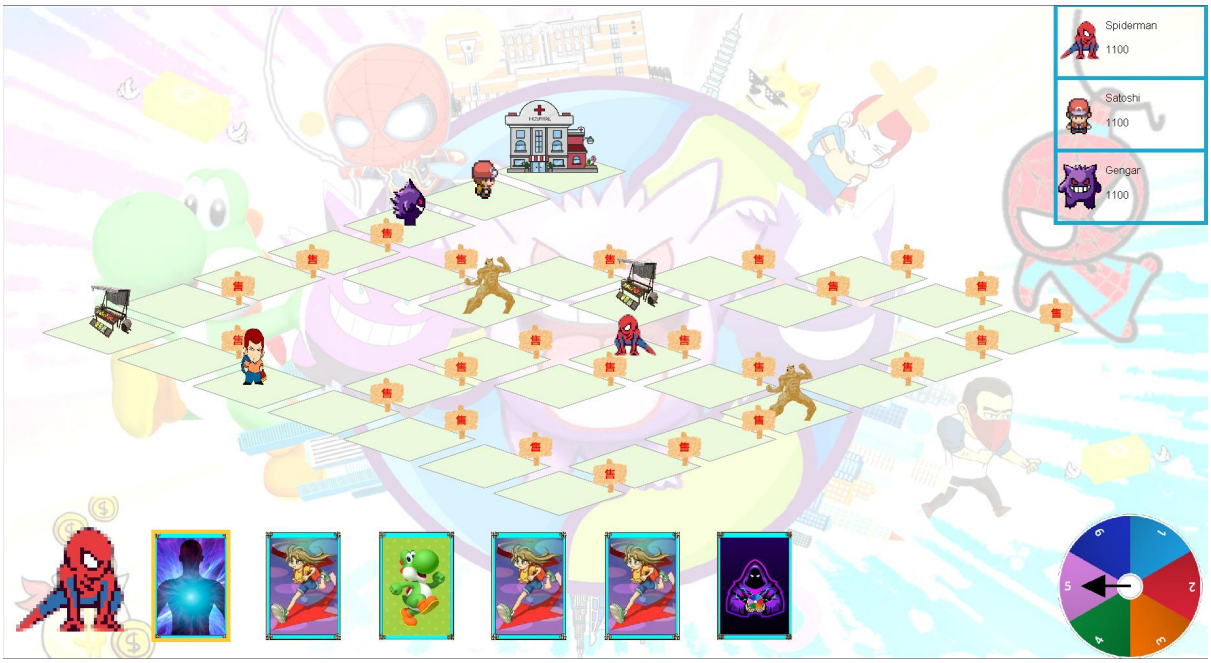
- Design your map!
  在我們的大富翁遊戲中，玩家可以自由的設計自己遊玩的地圖。

```
// available event and args of map:
// estate [price]
// thug [x,y](location of hospital)
// shop
// hospital
// dingo [x,y](location of hospital)
// null

START_OF_THE_DEFINITION
$ estate 1000
# thug 0 0
@ shop
% hospital
& dingo 0 0
^ null
END_OF_THE_DEFINITION

START_OF_THE_MAP
%^^$$$@
   $  $
  $&  #
 $@ $$$
$$ $$ $
$  $  $
$$$&$$$
END_OF_THE_MAP
```

如上面的文字，在 `assets/map` 底下新增或修改代表 map 的 txt 檔，依循合法的定義規則，就可以擁有玩家自定義的 event，舉例來說: `$ estate 1000` 就是指在地圖中 `$` 代表的是地產，而且他的基本價格是 1000。定義好了 event 與對應的符號，就可以來創建自己的地圖，創建地圖的方式就如同在 `START_OF_THE_MAP` 與 `END_OF_THE_MAP` 的符號排列方式一樣，下圖為這個地圖在遊戲中的樣子。



- User-frienly Interface
  在我們的遊戲畫面，玩家可以自由地縮放遊戲視窗，仍然可以維持正常的遊戲畫面。

Responsibilities of the team members

Overview

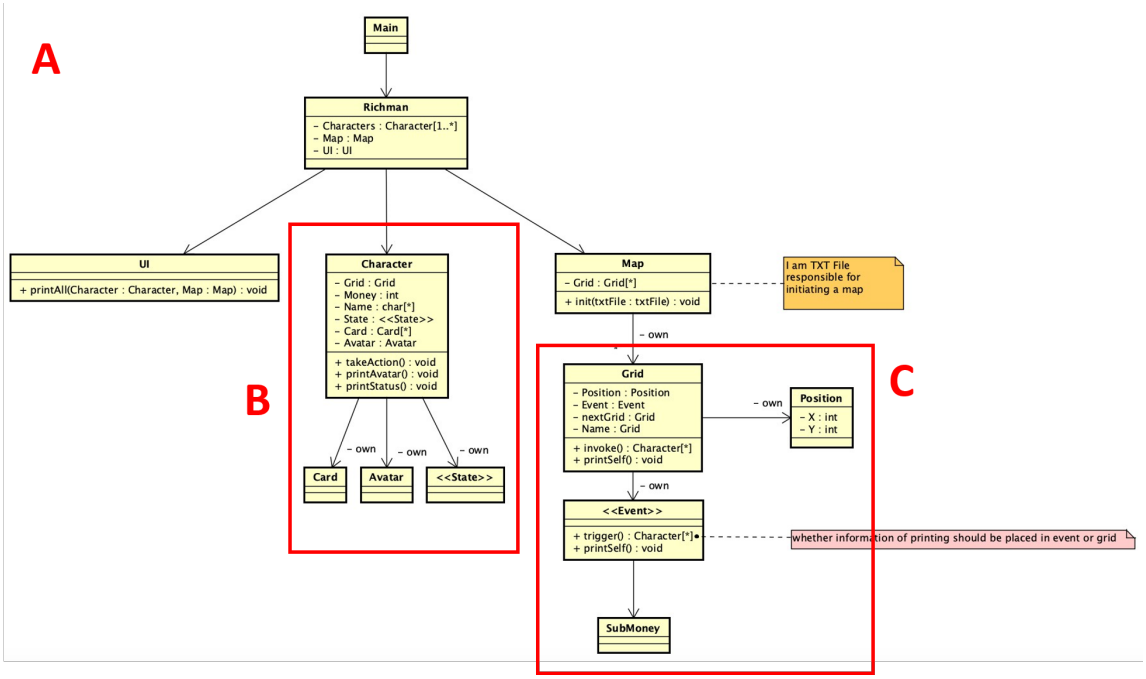
- 吳崇維: GUI, World, Event(Dingo, Shop)
- 邱譯: Role, Card, Mover, RoleState, Utils
- 張華恩: Grid, Event(Estate, Hospital, Thug)

Weekly meeting

我們每周會有一次的 meeting，討論當周情況以及下禮拜的進度，並依照當下的情況來決定分工內容。分配的原則是盡量不要讓職責重疊，大家可以獨立的開發而不受 conflict 影響

- Kickoff Meeting
  第一周決定了主題以及會有哪一些基礎的 object，也大致畫出 code 的骨架。將所有進度分成 4 個 phase，在 4 個禮拜中遵循進度完成。
  當時討論的 phases :
  - Phase 1
    能夠在顯示一個完整的簡單地圖，並且角色可以丟骰子前進。並顯示角色狀態列 (不需要有任何地產或是事件)完成完整的 Object 設計。
    主要希望先整合 UI / Model，能顯示最基本的遊戲畫面。
  - Phase 2
    補上方格屬性(地產和銀行)，並且設定地產升級以及角色跟地產的互動

- Phase 3
  基本卡片，商店等等相關設計
- Phase 4 (Improvement)
  UI優化(3D, 動畫)，更多卡片



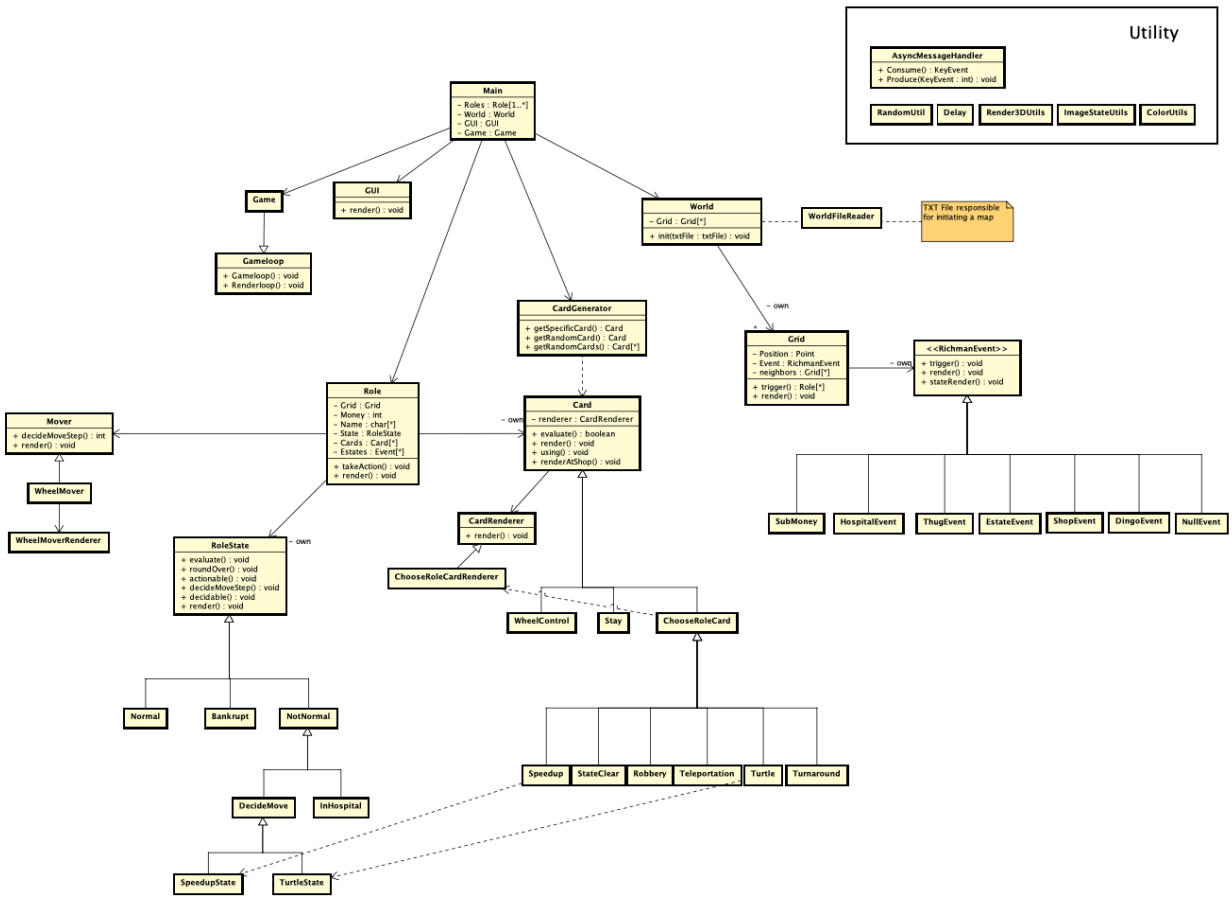
  這是 kickoff meeting 畫的初步 class diagram，以下是分工:
- A部分:吳崇維
- B部分:邱譯
- C部分:張華恩

- Week 1
  當周完成了預定的進度，也發現不少延伸問題，將問題與預定進度整理出來
  - grid 的實作
  - 鍵盤監聽的 pipe 實作
  - 實作地產 event
  - 可以 render 出角色的屬性
  - 基本卡片設計
- Week 2
  完成了當周進度，討論下禮拜的工作分配
  - 基本卡片，商店等等相關設計
  - event
    - 地產 (3種地價，每一種地價有0 / 1 / 2 / 3 三種level)
    - 野狗 (hospital)
    - 暴走族 (hospital)
    - 商店
  - nameToEventMapper: basicMap.txt 可以設置相關參數
  - 更多卡片
- Week 3
  - 完善房地產相關
  - 卡片商店設計

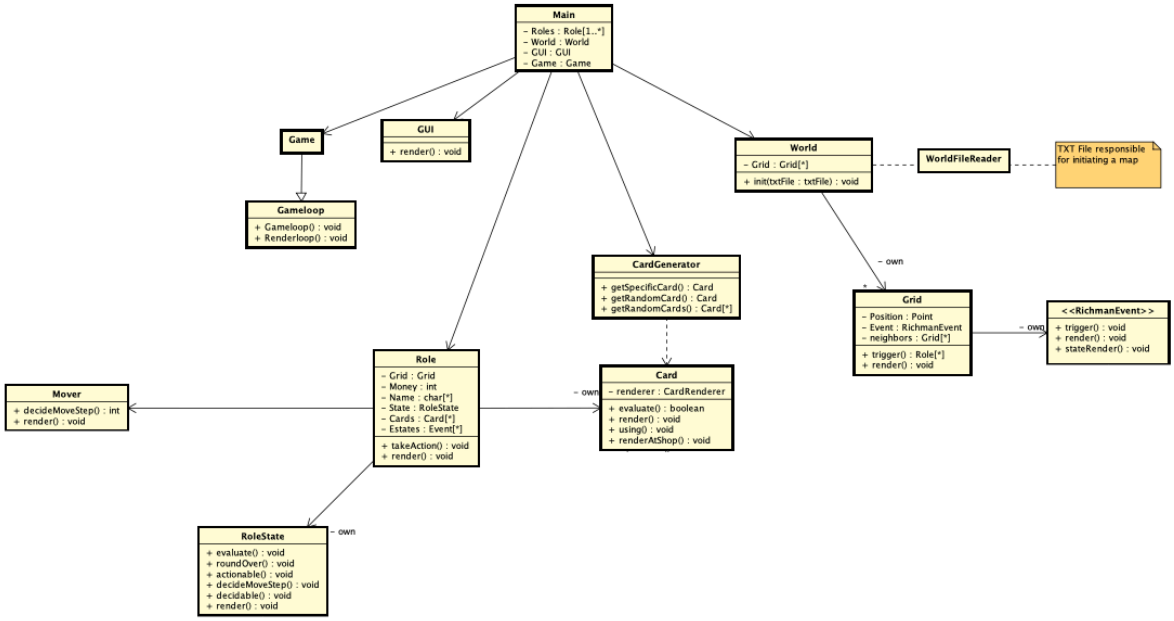
    - 卡片補全
    - 整體版面改進以及 3D 配置
- Week 4
    - Final Report
    - Bug Fixed
    - 細節修正

Class Design (Advantage and Disadvantage)

Class Diagram



Overall



我們的程式流程由`RenderLoop`以及`GameLoop`組成。
`GameLoop`負責管控每個角色的行動(包括移動，購買...等等所有行為)的邏輯以及順序先後。
`RenderLoop`負責將每一個 object 呈現在畫面上。

- 初始化
  初始化圍繞在代表 Map 的 txt file，如上述 "How to play Rich Man" 所提到，我們根據map file 的資訊來決定我們應該 new 出哪些物件，以及應該如何初始化物件。
- 如何 Render
  所有真正會在畫面出現的 object (角色、卡片、房地產、事件) 等等，都可以從`World`這個class 中的 attribute 遞迴找到。`World`可以 access 到所有實際出現的 objects。
  所有的 object 必須自己知道該如何 render 自己，也就是實作render()。`RenderLoop`每隔毫秒會去要求`World`更新並 render 畫面，`World`便會往下發布 render 的指令，遞迴地將所有 object render出來
- GameFlow
  遊戲邏輯流程十分單純，我們只需要一個 loop 去循環角色的行動

```
while (running) {
    for (Role role : roles) {
        if(role.isAlive()){
            role.takeAction();
        }
    }
}
```

我們只要確保初始化的正確，以及每一個 action 中的邏輯正確，可以維護每一個 object 合法的運作，就可以降低程式流程出錯機率。
也因為此遊戲的操作不會接收同時多人的指令(即一個指令一個動作)，GameLoop 不用擔心任何 race condition，只需專注當下的 action 正確與否，這也大大降低程式碼的複雜度。

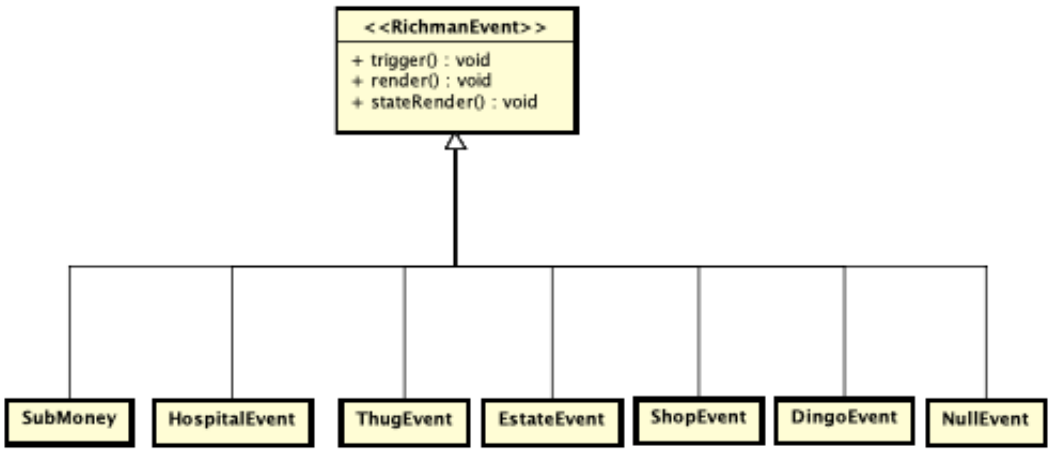
Open–Closed Principle 設計

我們在卡片、方格事件、角色人物這些常需要橫向擴充的 class 遵守 OCP 原則，來確保有新的 class 加入/移除的時候不會修改到整體架構，進而增加程式碼的可擴充性及易維護性。

- 如何達成
  將目標系列 class 的共同特性抽取出來，寫成一個 general 的 interface，並在 Main 用一個 interface 的 list 來記錄現在共有哪些可用的 implementation。每當系列 class 要增減，就可以在寫完 class 的實作後直接在該 list 中管理。
- 實際應用
  例如我們想要增加一張停留卡，我們便在`src/model/card`底下新增一個`Stay.java`的 class。接著在 Main 中管理卡片 interface 的 list 中新增該 class。

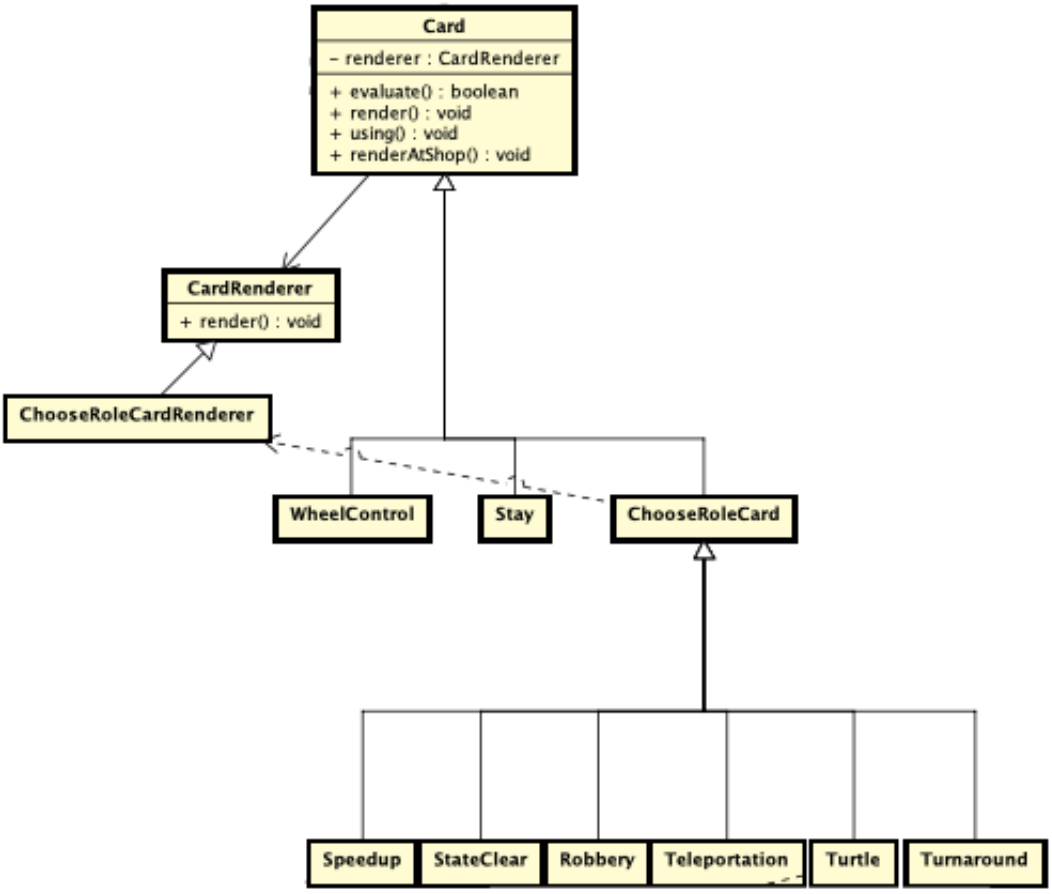
```
// Main.java L56
private static HashMap<String, Card> cardConfig() {
    HashMap<String, Card> nameToCardMapper = new HashMap<>();
    nameToCardMapper.put("stay", new Stay()); // 在此做增減
    nameToCardMapper.put("turnaround", new Turnaround());
    ...
    nameToCardMapper.put("speedup", new Speedup());
    return nameToCardMapper;
}
```

Event



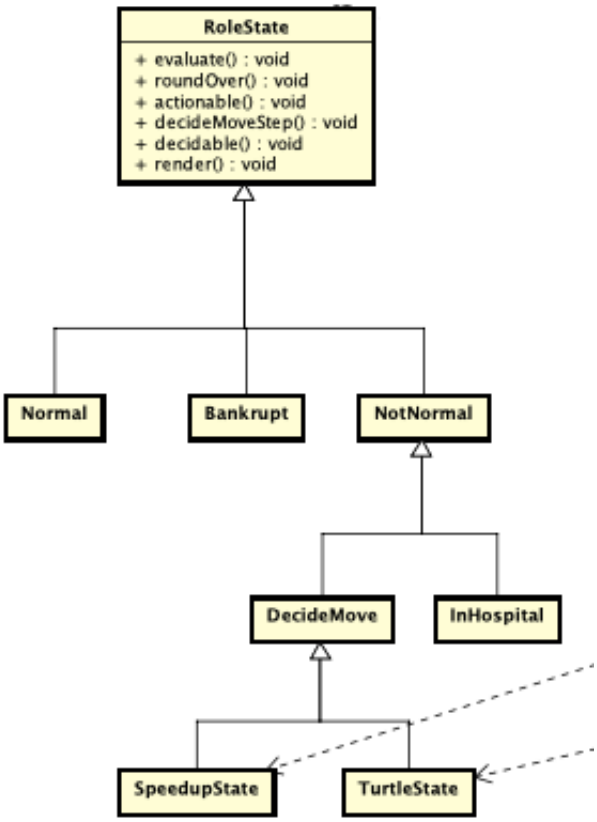
我們設計了一個`RichmanEvent`的 interface，定義了每個 event都會有的行為，並且使用一個 abstract class，稱為`RichmanEventAbstract`去實作`RichmanEvent`與`Cloneable`(因為一開始每一種 event只有 new 出來一個，但是屬於不同格子的同一種 event 必須是 deep copy，才不會在修改時所有格子裡的 event都一起變化)。最後，所有的 event 都會去 extends這個`RichmanEventAbstract`。

Card



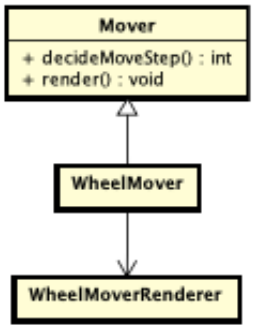
我們先設計了一個`Card` abstract class，定義好一張卡片需要有哪些行為，比較特別的是，不同種類的卡片常常會有類似的行為，例如許多卡片都需要選擇對象，因此我們也設計了不同的 abstract class 描述這些共同的行為，例如 `ChooseRoleCard`，這樣在開發時能夠避免重複的程式碼，不僅更快速也減少錯誤。

RoleState



我們先設計了一個`RoleState` abstract class，定義好一個狀態需要定義哪些行為，與卡片相同的是，不同種類的狀態常常會有類似的行為，例如許多狀態會決定角色移動的步數，因此我們也設計了不同的 abstract class 描述這些共同的行為，例如 `DesideMove`。

Mover



我們先設計了一個`Mover` abstract class，當希望能加入不同方式來控制角色行動時，僅需要繼承這個 abstract class，實作需要的函示，即可再不更改其他程式碼的狀態下加入新的 Mover。

統合鍵盤輸入以及讀取的管道

由於我們的程式運行中會分成不同職責的 thread，所有聆聽鍵盤的需求也散落在各處。為了統一所有鍵盤聆聽的需求，我們有一個溝通 class 負責這件事

```
public class AsyncMessageHandler {
    BlockingQueue<KeyEvent> blockingQueue = new LinkedBlockingDeque<KeyEvent>
();

    public KeyEvent consume() throws InterruptedException {
        return blockingQueue.take();
    }

    public void produce(KeyEvent keyEvent) {
        blockingQueue.add(keyEvent);
    }

    public void clear() {
        blockingQueue.clear();
    }
}
```

因此所有需要鍵盤指令的地方都會 call 同一個 consume 函式，進而維護每個 input 的觸發先後邏輯。呼叫 clear 函式，來確保在跑動畫過程(並非輸入階段)中輸入的指令不會被隨意讀取。

User-frienly Interface

在畫面 render 時，我們並非直接選定座標來 render，而是計算這個物件應該在整個視窗的哪個位置，這樣可以確保玩家有不同大小的視窗時，也能夠有完全一樣的遊戲畫面。

Other packages that you have used

None