Final Report

Game HearthClone

Team Members and Responsibilities

b07902040 吳承軒：Client-Server網路資料傳輸
b07902042 葉璟諄：架構設計、Game及卡牌實作、Report
b07902052 鄭達詠：架構設計、View及Controller實作、Report

Abstract

因為我們幾年前都有玩過**HearthStone 爐石**這款遊戲，但近年來他的卡牌越出越多，找不到當初的樂趣，因此就希望能參考爐石的遊戲架構，自己寫出一個記憶中美好版本的爐石，剛好在做HW1 HW2的時候就有一些想法，決定在Final project把它實作出來，取名為**HearthClone**，因為是卡牌對戰遊戲，所以我們希望能做出網路連線功能，才不會看到彼此的手牌。

遊戲內容大概是兩個人的回合制卡牌對戰遊戲，每個人會有牌庫(Deck)，會從其中抽出卡牌到手牌兩個人的回合輪流，自己的回合可以消耗法力(Mana)打出卡牌，卡牌分為手下(Minion)和法術(Spell)兩種類型，手下有消耗(Cost)，生命值(HP)和攻擊力(ATK)，打出後可能會觸發特殊效果，並且占用場上的一個位置，可以進行攻擊等動作，而法術通常是即時性的效果，可能有特定的目標也可能是範圍效果(AOE)，遊戲目標是摧毀敵方的英雄(Hero)，英雄有生命值，用手下或法術攻擊就可以造成傷害。

Game Design

我們使用MVC的架構來實作這個遊戲，主要分成Model(Game)、View、Controller分別運作，透過EventManager傳送Event通知來達到同步溝通。這個遊戲可以連線進行，分成一個Server端跟兩個Client端，其中Model由Server端管理，而Clinet端新增一個Gameinfo來記錄目前遊戲的進程，並由View跟Controller來控制顯示介面跟操作輸入。

`package event`

`interface EventManager`

由`ClientEventManager`、`ServerEventManager`、`LocalEventManager`實作，用來同步遊戲各部分的核心。當遊戲中發生改變需要通知其他部門，則負責的單位會向`EventManager`提出(`post`)一個`Event`，而由`Event Manager`去通知(`notify`)各個部分這個事件發生來達到同步的效果。

`interface EventListener`

當`Event`發生時`EventManager`會通知的對象，`Game`、`View`、`GameInfo`皆有實作。實作`EventListener`的class需要override `notify()`。

`interface Event`

`EventManager`向`EventListener`傳達用的訊息，`Game`、`View`、`Controller`在遊戲發生變化時向`EventManager`提出，再轉達給所有有`regeister`的`EventListener`，由`EventTurnStart`、`EventCardClicked`、`EventGameEnd`等class實作。

`package model`

`class Game implements EventListener`

遊戲的本體，負責管理game loop，判斷遊戲邏輯，以及當前的遊戲狀態。`run()`不中斷的傳送`EventEveryTick`來推動遊戲的更新。`Game`的properties中有他所`register`的`EventManager`、使用的`Deckloader`、管理狀態的`StateMachine`、保存目前兩位玩家的`List<Player>`、以及保存遊戲進度的一些參數，有`Player`、`Minion`、`Card`等不同Type。

`class GameInfo implements EventListener`

Client端的遊戲紀錄，可以視為`Game`的複製體，但僅保存`View`跟`Controller`所必需的資訊。本身接收Event來更新使自身與`Game`一致，但不改變遊戲狀態。

`abstract class DeckLoader`

負責載入玩家所使用的套牌。其中有`library`存放所有可以抽到的卡牌。延伸`DeckLoader`的class可以override `loadDeck()`，根據自己設定的規則，隨機切出兩副牌，並回傳`ArrayList<ArrayList<Card>>`裡面儲存兩個玩家的`Deck`。

`class StateMachine`

負責管理遊戲狀態，主要結構是stack，由內部的一個`ArrayList<Integer>`實作，最上層的是目前的遊戲狀態，由定義在`Const`的數字表示各個狀態。

`class Player`

負責管理一個玩家的當前資訊，以及可以進行的操作。`Player`的properties中包含他上層所屬的`Game`、分別表示牌庫(deck)跟手牌(handCards)的兩個`List<Card>`、分別表示自己及對方場上手下的`List<Minion>`

`interface Card`

遊戲內部所有卡牌皆實作這個interface，`Minion`、`Spell`兩個interface皆有繼承。需要實作`getName()`、`getCost()`、`getCost()`等method來獲取該卡片的資訊。

`interface Minion extends Card`

遊戲內部的所有手下卡都實作這個interface。需要實作`getHP()`、`getATK()`、`isAlive()`等獲取手下資訊的method，或是`attack()`、`doTurnEnd()`等定義特定動作的方法。

`abstract class AbstractMinion implements Minion`

實作`Minion`，遊戲內部的手下卡都繼承它。簡單實作`Minion`中的method及定義參數。

`interface Spell extends Card`

遊戲內部的所有法術卡都實作這個interface。需要實作`takeEffect()`來定義法術觸發時的效果。

`abstract class AbstractSpell implements Spell`

實作`Spell`，遊戲內部的手下卡都繼承它。簡單實作`Spell`中的`takeEffect()`。

`interface Targeting`

由部分手下或法術實作的interface，會去override `getCandidate()`並根據自己的效果篩選目標

`interface BattleCry`

由部分手下卡實作的interface。打出有`BattleCry`的Minion時會觸發`doBattleCryEffect()`，內容由實作`BattleCry`的`Minion` Override.

`interface DeathRattle`

由部分手下卡實作的interface。有`DeathRattle`的`Minion`死亡時會觸發`doDeathRattleEffect()`，內容由實作`DeathRattle`的`Minion` Override.

`interface Taunt`

由部分手下卡實作的interface。`Minion`必須優先攻擊`Taunt`的Minion。

`interface Poisonous`

由部分手下卡實作的interface。受到`Poisonous`的`Minion`傷害的目標，就會直接死亡。

`interface DivineShield`

由部分手下卡實作的interface，有`DivineShield`的`Minion`第一次受到傷害會消耗掉`DivineShield`並且免疫傷害。

`interface Stealth`

由部分手下卡實作的interface，有`Stealth`的`Minion`在第一次攻擊前是無法被攻擊與`Targeting`的。

`interface Charge`

由部分手下卡實作的interface，有`Charge`的`Minion`進場的回合就可以攻擊(一般`Minion`要下一回合)

`interface WindFury`

由部分手下卡實作的interface，有`WindFury`的`Minion`一回合有兩次攻擊機會。

`interface Combo`

由部分手下卡實作的interface，有實作`Combo`的`Minion`會知道自己是這回合打出的第幾張牌，並觸發特別效果。

`package view`

`class View implements EventListener`

負責控制GUI輸出。每個tick在收到`EventEveryTick`的時候根據目前的`GameInfo`更新畫面一次。`View`的properties包括他對應到的`EventManger`、聯動的`Controller`、`GameInfo`、內部繪畫的物件`List<Painter>`、動畫列`List<Animation>`、以及其他用來記錄狀態的圖像跟參數，包含Animation等類別。

`interface Painter`

負責繪出指定的圖像。實作`Painter`的class需要override `draw()`。`View`維護目前有的Painter，並在更新時呼叫每個`Painter`的`draw()`。

`interface Animation extends Painter`

繼承自Painter負責繪出動態圖像。實作`Animation`的class需要多override `isExpired()`、`update()`等獲取動畫進度或操作動畫的method。`View`維護目前有的`Animation`，並在更新時移除過期的`Animation`。

`package controller`

`class Contoller extends MouseAdapter implements EventListener`

負責管理來自玩家的輸入並送出相關的`Event`。`Controller`被登記在`View`的`JPanel`底下接收`MouseEvent`，並根據座標跟`GameInfo`判斷點擊對應的操作並向`EventManager`送出`Event`。

`package client`

`class Client`

負責處理client端的進行，以及訊息接送和輸出。在`start()`中創建這一端的`ClientEventManager`、`View`、`Controller`跟`GameInfo`，等待來自`Server`的訊息並開始遊戲。

`package server`

`class Server`

負責處理server端的進行，以及訊息接送和輸出。在`start()`中創建`ServerEventManager`跟`Game`，向`Client`發送訊息並等待回應，接上後呼叫`Game.run()`開始遊戲進行。

Advantage

1. **工作分配**
   MVC的遊戲架構下Model、View、Controller分工清楚，開發的過程中容易分工，不會互相干擾。
2. **卡牌擴充**
   卡牌的設計上，大致上符合了OCP，因此如果想要生成一張新卡牌只要使用static把一些property設定好，並且根據我們的規則進行實作與override，就可以很迅速的生成一張卡，我們總共有120張卡，擴充時只要參照行為相似的卡片，基本上不會出bug。
3. **繪圖物件擴充**
   Painter及Animation皆用物件的形式表達，符合OCP，如果需要新增動畫或是場上的物件，只需要增加一個對應的Painter/Animation並在View中創建instnace即可，相當容易擴充，例如未來可以為卡片很輕易的增加效果動畫。
4. **組牌方式擴充**
   DeckLoader為一個可以被繼承的abstract class，需要不同的組牌方式只需另外創建一個繼承他的子類別並override loadDeck()即可。未來可以為遊戲新增選牌介面。
5. **效果擴充**
   Deathrattle、BattleCry、DivineShield等效果分類是以interface的形式表示，手下可以輕易地兼具不同效果。且未來若要新增特殊的效果(例如life steal、rage等爐石有的關鍵字)也可以簡單實裝，不須為其他minion新增參數。

Disadvantage

1. **與遊戲事件相關的效果**
   對於遊戲流程的處理我們皆由條件判斷，沒有做物件的包裝，因此如果要添加在特定條件下觸發的特殊能力(例如：有人抽牌時+1/+1、對方手下攻擊時抽一張牌)勢必會需要在遊戲流程增加判斷，無法輕鬆進行擴充。
2. **client-server同步問題**
   在Local執行遊戲跟透過連線執行遊戲分別可能產生不同的同步問題。
   在local執行遊戲時，由於Event的post跟notify皆是由function call執行，假設今天有A、B兩個listener要接收一個Event e1，但在notify A的過程中A又post一個新的Event e2，重新notify所有listener，如此重複循環下去，可能經過多個function最後才能notify e1給B，而當下的state不能保證是否跟當初post e1時預期的相同。需要對state跟Event的連動有很精確的掌握跟設計才能避免產生問題。
   透過連線執行遊戲時，由不同端post出來的Event是由網路傳送，而負責等待訊息並接收的是一個獨立的thread，無法預期何時會收到並notify。假設今天View根據當下的state判斷，確定要執行一段程式，但在執行之前收到Event並改變了state，而View繼續執行該段code，就有可能產生synchronization的問題(例如Event處理移除了View本來想畫的卡片)。

Other Packages

GUI

- java.swing
- java.awt

卡牌圖片

- Hearthstone的卡片圖片與設計
  https://playhearthstone.com/en-us/cards
- pixel-me
  https://pixel-me.tokyo/en/

背景音樂

- CodeJava Written by Nam Ha Minh:How to play back audio in Java with examples
  https://mail.codejava.net/coding/how-to-play-back-audio-in-java-with-examples
- Hearthstone Soundtrack - Main Title
  https://www.youtube.com/watch?v=WXxy3Y8daks&ab_channel=HearthPwn

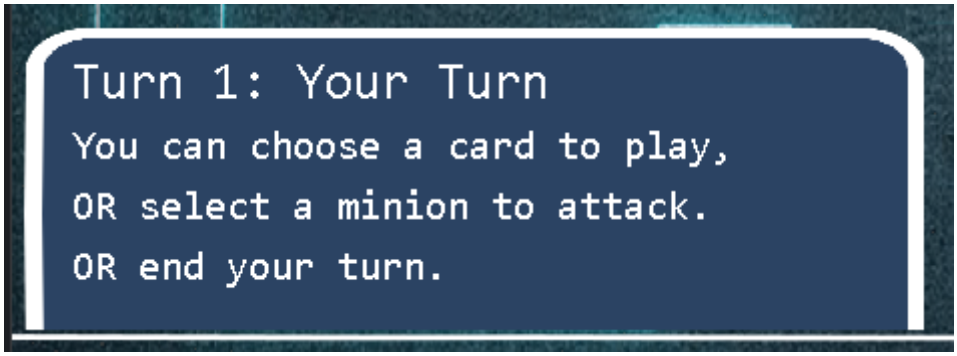
How to Play

執行程式

- local端開兩個視窗

  ```
  bash runLocal.sh
  ```

  起始兩個視窗會疊在一起，由左下顯示Your Turn的視窗開始操作。

- server - 2 clients
  先開Server，再開Clients
  Server：在工作站linux3上 server預設開port 2021，執行下列指令

  ```
  bash runServer.sh
  ```

  Client：分別在兩台機器，預設輸入linux3 ip = 140.112.30.34，port = 2021

  ```
  bash runClient.sh $(ip) $(port)
  ```

操作順序

只有在自己的回合能進行操作，操作都是遵照這個順序：

1. 點擊手牌或手下可以放大查看效果，當下可以使用或攻擊的卡會發光，並且放大後的select處於可點擊的狀態，如果想換張卡片，可以直接點別的卡片來放大，或是點空白處取消放大
2. 放大狀態下select即可選定使用這張卡片，有些select後會直接施放，若是需要指定目標就需要再選定一個目標，同樣是放大後點下select
3. 你的回合結束後按下Endturn就會換到對手的回合

規則

1. 目標擊敗敵方英雄
2. 回合開始會抽一張牌，手牌最多只有十張，超過的話抽取的牌會直接消失，不會進入手牌
3. 起始有一顆水晶，之後每個你的回合會獲得一顆水晶，並且補滿全部水晶
4. 打出卡片後會根據卡片花費消耗水晶，若是手下就會召喚到場上(上限7隻)
5. BattleCry是指打出該手下會觸發的效果，有些需要指定目標

6. DeathRattle是該手下死亡後會觸發的效果
7. 手下攻擊時會對目標造成自己ATK的傷害，自己受到目標ATK的傷害
8. 手下一回合只能攻擊一次(WindFury的腳色可以攻擊兩次)
9. 手下被召喚的回合不能攻擊(Charge的手下可以在召喚那回合攻擊)
10. 要優先攻擊有Taunt的手下(有盾牌的手下)
11. Poisonous可以直接摧毀傷害到的手下
12. DivineShield可以抵擋第一次受到的傷害，隨後消失
13. Combo是指本回合打這張卡前要先打過別張卡，才會觸發效果
14. 牌庫抽完後每抽一張牌會受到累加的疲勞傷害