國立交通大學

資訊科學與工程研究所

碩 士 論 文

魔法牌遊戲開發系統之研究

The Study of Trading Card Game Development System

研 究 生：陳威霖

指導教授：吳毅成　教授

中 華 民 國　九 十 七　年 七 月

魔法牌遊戲開發系統之研究

The Study of Trading Card Game Development System

研 究 生：陳威霖　　Student：Wei-Lin Chen

指導教授：吳毅成　　Advisor：I-Chen Wu

國 立 交 通 大 學

資 訊 科 學 與 工 程 研 究 所

碩 士 論 文

A Thesis
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in

Computer Science

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

魔法牌遊戲開發系統之研究

研究生：陳威霖　　指導教授：吳毅成

國立交通大學 資訊科學與工程研究所

摘要

魔法牌遊戲是一種遊戲類別。本論文提出了魔法牌的明確定義與系統架構，並且實際做出了一個可以開發魔法牌遊戲的魔法牌遊戲開發系統。這個系統以 Java 語言為基礎，透過讀取指令集的方式包裝程式碼，亦允許遊戲設計師自行寫擴充的函式供系統使用。我們所提出的系統，就像是一種包裝在某種程式語言(在此使用 Java)的程式語言一般，故透過該系統，遊戲設計師可以製作各式各樣的魔法牌遊戲。

The Study of
Trading Card Game Development System

Student: Wei-Li Chen Advisor: I-Chen Wu

Institute of Computer Science and Engieering
National Chiao Tung University

Abstract

TCG(Trading Card Game) is one of the most popular card games. In this thesis we analyze TCG and derive a rigorous definition and structure of TCG. Furthermore, we also establish a practical system to develop TCG games. Based on Java programming language, we package program code by reading instruction in this system. Game designers also have permit to write extensive function for system usage. Game designers can make varieties of TCG through this system.

誌謝

首先，我要感謝上帝，讓我在大三的時候就對魔法牌這個主題產生興趣，因而從大學的專題就開始做這種遊戲的相關研究，以致於我有比別人更多的時間來熟悉我的題目與做我的研究。並且也能夠在研究所繼續這個題目的研究。

我也要感謝在這整個研究過程中，對我感到放心、信任，並且極富耐心，總是願意給我建議的指導教授吳毅成教授。謝謝老師總是接納我在表達上的不完全，雖然許多時候我們的討論常讓我感受到挫敗或是沒有進展，但每次只要想到老師願意花時間繼續和我討論與給指引我改進的方向，就覺得很感動。

我也要感謝博士班的汪益賢學長，雖然碩二這一年學長不在交大，但從大三我接觸這個題目開始，學長就教我提供我許多寶貴的意見，甚至學長也常是我在寫程式遇到瓶頸時的救命恩人，所以，在此我也要特別感謝汪益賢學長。

我也感謝口試委員們的批評與指教，有許多的意見對我而言是非常寶貴的

感謝實驗室的每一位同學，大家雖然都忙碌，不過仍然可以有彼此相顧的機會，也謝謝大家最後這段時間的陪伴與一起奮鬥。

感謝團契裡每一位為我禱告與打氣的弟兄姊妹們，有你們精神上的支持與同在，是我不斷前進的動力。

最後，感謝我的父母，在我求學生涯裡，總是給我許多實際的幫助，謹以此論文，獻給每位支持我的人。

目錄

摘要 i
誌謝 iii
目錄 iv
表目錄 vi
圖目錄 vii
第一章、緒論 1
1.1 魔法牌遊戲(TCG)介紹 1
1.2 範例魔法牌－「99 魔法牌」介紹 2
1.3 研究目標 5
1.4 論文章節介紹 5
第二章、系統架構 6
2.1 魔法牌遊戲之定義 6
2.2 魔法牌遊戲系統架構 7
2.2.1 物件化 7
2.2.2 階段系統 10
2.2.3 動作系統 11
2.2.3.1 動作系統的特性 11
2.2.3.2 動作系統的設計 15
2.3 魔法牌遊戲開發系統設計 18
第三章、魔法牌遊戲系統實作 19
3.1 系統概要 19
3.2 遊戲基本物件 20
3.3 階段流程 23
3.4 動作系統 25
3.5 其他底層的重要設計 30
3.6 魔法牌遊戲開發系統之設計 31
第四章、使用本系統開發魔法牌遊戲 32
4.1 系統環境簡介 32
4.2 魔法牌遊戲開發程序 32
4.3 遊戲基本設定 33
4.4 物件化類別定義 34
4.5 階段流程設定 35
4.6 物件與動作設定 37
4.7 遊戲擴充 39

第五章、結論與未來展望 40
附錄 A 執行條件與執行效果的文法 41
附錄 B 「99 魔法牌」物件設定 50
附錄 C 其他程式相關說明文件 57
參考文獻 65

表目錄

表 1：「99 魔法牌」的牌卡功能列表 4
表 2：「99 魔法牌」的物件類別屬性值一覽表 9
表 3：「99 魔法牌」各種牌卡的動作一覽表 13
表 4：遊戲(Game)類別的詳細說明 20
表 5：玩家(Player)類別的詳細說明 21
表 6：牌區(Area)類別的詳細說明 22
表 7：牌卡(Card)類別的詳細說明 23
表 8：階段(Stage)類別的詳細說明 23
表 9：動作(Action)類別的詳細說明 25
表 10：執行時機(Specification)類別的詳細說明 26
表 11：執行時機設定一覽表 27
表 12：執行條件(Condition)類別的詳細說明 28
表 13：執行條件設定一覽表 28
表 14：執行效果(Operation)類別的詳細說明 29
表 15：執行效果設定一覽表 29

圖目錄

圖 1：魔法牌遊戲樹狀物件架構 7
圖 2：魔法牌遊戲物件關係圖 10
圖 3：「99 魔法牌」的階段流程 11
圖 4：「99 魔法牌」裡的 10、Q、Joker 14
圖 5：魔法牌主要物件的 class diagram 19
圖 6：動作系統相關設定的 class diagram 24
圖 7：遊戲基本設定畫面 34
圖 8：物件化類別設定畫面 35
圖 9：階段設定畫面 37
圖 10：物件與動作設定畫面 38
圖 11：使用文法(grammar)設定執行效果畫面 39

第一章、緒論

本章將簡介本論文所需要的背景知識，然後提出本論文的研究目標，並且會在最後簡介本論文的內容大綱。在 1.1 節裡，我們會介紹魔法牌這類型遊戲，1.2 節用一個簡單的範例遊戲幫助讀者了解魔法牌的特性。1.3 節提出我們的研究目標。1.4 節則對本論文做個簡要的章節介紹。

1.1 魔法牌遊戲(TCG)介紹

集換式卡片遊戲(Trading Card Game, 簡稱 TCG)[1]，是指使用特殊的遊戲專用牌卡所進行的卡片遊戲。這類型遊戲一方面在市面上以實體牌卡的方式販售與推行，另一方面也透過網路更快速的推廣。

魔法牌遊戲與一般的卡片遊戲最大的差別在於：一種魔法牌遊戲裡通常會有上百種以上的牌卡，玩家可根據規則自由地從自己所擁有的牌卡中選出一定數量的牌並組成「牌組」(Deck)，再以「牌組」來進行遊戲。遊戲的玩法會依玩家所使用的牌組而有所不同，所以不同的玩家或不同的牌組之間就可以構成非常多元的遊戲組合，使得魔法牌遊戲玩起來非常富變化性。除此之外，魔法牌遊戲設計者還可不斷設計發售新的牌卡，使得一款魔法牌遊戲的組合與變化更加的多樣性。因為這類型遊戲每次玩起來時規則都有如魔法般千變萬化，故又稱為「魔法牌」，亦有人以「戰牌」、「鬥牌」稱之。

魔法牌遊戲的牌卡本身除了具備「遊戲性」（可以玩）之外，同時也具備「蒐集性」。玩家可以透過各種交易行為（向販售商購買、與人交易或交換、參與比賽賺取新牌卡等等）來取得牌卡，隨著自己所擁有的牌卡數量的增多，也會改變自己進行遊戲時的強度，所以在「遊戲」與「蒐集」的交互作用之下，對玩家具有強大的吸引力。

第一款魔法牌遊戲「魔法風雲會」(Magic: The Gathering, MTG)[2]是 1993 年美國數學家 Richard Garfield 所設計，然後由威世智(Wizards of the Coast)公司販售，據官方統計，目前至少有十種語言的版本，全球有超過 600 萬名玩家，並且有超過 9000 多種不同的牌卡被印刷。日本漫畫「遊戲王」[3]由漫畫家高橋和希於 1996 年開始創作，內容亦以魔法牌遊戲的遊戲型式為核心戰鬥方式，在日本也頗為有名。因「遊戲王」漫畫的名氣，「遊戲王」隨後也推出魔法牌遊戲實體卡在市面上販賣。

隨著網路的發達，許多魔法牌遊戲開始網路化，「魔法風雲會」推出了「魔法風雲會網路版」(Magic Online)[4]，而「遊戲王」也推出「遊戲王網路版」（YuGiOh online）[5]。隨後，亦開始有許多的故事被以魔法牌遊戲的型式包裝，以實體卡或是網路遊戲的方式出現在市面上。譬如線上「遊戲仙境傳說」(RO)[6]、「魔獸世界」(WOW)[7]在運作成功之後，相繼推出魔法牌遊戲版本[8][9]，以吸引更多的玩家。日本遊戲公司 KOEI 於 2000 年開始推出的「真・三國無雙」[10]在電玩主機 Play Station 2(PS2)上受到玩家熱列迴響，之後相繼推出十餘款相關遊戲，亦於 2007 年看好魔法牌遊戲市場，進而推出「真・三國無雙」實體卡[11]，進軍魔法牌遊戲市場。其他如日本知名卡通「鋼彈」[12]、「神奇寶貝」[13]、或是中國文化的五行星宿[14]]也亦被包裝成魔法牌遊戲發售過。

1.2 範例魔法牌－「99 魔法牌」介紹

因為一般的魔法牌規則都極為複雜，所以在此我們自行設計一套簡單的魔法牌範例遊戲－「99 魔法牌」，希望透過介紹這個遊戲可以幫助讀者對魔法牌遊戲有更多的認識。

「99 魔法牌」的遊戲規則與撲克牌遊戲裡的「99」很像，規則如下：

1. 遊戲使用遊戲專用牌進行遊戲，這些牌卡和撲克牌相似，上頭分別有 1~10, J, Q, K 與 Joker 的符號，牌上附有一些文字的規則敘述(表 1)。玩家在進行遊戲時，需要遵循牌上的規則敘述。
2. 每位玩家擁有自己的牌。
3. 玩家依規則每次只能選擇自己一部份的牌，組成牌組(Deck)進行遊戲，規則為：
   - 牌組的牌數限定 40 張。
   - 相同數字的牌最多只能帶 4 張。
   - 「特殊牌」最多只能帶 16 張(「特殊牌」會在牌上有所標記)
4. 兩個人進行遊戲。
5. 遊戲開始後，每位玩家先將自己的牌組洗牌，然後抽五張到自己手上，其他的牌以面朝下的型式放在自己的前面。這些面朝下的牌，總稱之為「牌庫」。
6. 所有玩家只能看自己手上的牌和所有玩家用過的牌。
7. 隨機選擇一個人，從這個人開始，依順時針方向依序做下面的動作：
   - 出牌（輪到的玩家從自己手上挑一張牌出到桌面上）
   - 抽牌（輪到的玩家從自己牌庫抽一張牌到自己手上）
8. 遊戲裡有一個遊戲積數，一開始為 0，玩家出牌時，遊戲積數會根據牌上的規則而改變。牌的規則如表 1。
9. 遊戲積數不得超過 99，輪到的玩家沒牌出時，該玩家就算輸。最後一位留下來的玩家是贏家。
10. 遊戲的贏家可以從最先輸的輸家的牌庫中挑一張牌做為戰利品。
11. 遊戲設計者可以設計新牌以調整遊戲的平衡性與增加遊戲的趣味性。

牌	功能	特殊牌
A	遊戲積數+1	否
2	遊戲積數+2	否
3	遊戲積數+3	否
4	遊戲積數+4	否
5	遊戲積數+5	否
6	遊戲積數+6	否
7	遊戲積數+7	否
8	遊戲積數+8	否
9	遊戲積數+9	否
10	遊戲積數+10 or 遊戲積數-10	是
J	Pass	是
Q	遊戲積數+20 or 遊戲積數-20	是
K	遊戲積=99	是
Joker1	別人用減之後，強迫他從手上隨機棄掉一張牌	是
Joker2	別人用減之前，先將遊戲積數-5 然後強迫他再出一張牌	是

表 1：「99 魔法牌」的牌卡功能列表

因為玩家每次只能帶 40 張牌進入遊戲，而牌總共有 14 種（當然還可以更多），相同數字的牌最多只能帶 4 張，具有特殊功能的特殊牌又有 16 張的限制，所以每個玩家在選擇所要使用的牌組時會因其考量的不同而有所差異。因為遊戲設計者還可以設計發行新牌，所以玩家組牌的選擇會越來越多，整個遊戲玩起來的變化性也就會越來越大。

玩家從總數量會越來越多的牌裡挑選部份自己喜歡的牌來進行遊戲，這就是魔法牌遊戲最大的特色！

1.3 研究目標

魔法牌遊戲是一種遊戲類型，該類型的遊戲也不斷地在發展與增加當中，不同的魔法牌遊戲之間雖然在規則上有許多的差異，仍然有些共通的性質是可以整起來的，但至今並沒有人針對這種類型的遊戲做整合與分析，我的目標就是希望可以透過分析這些魔法牌遊戲，提出魔法牌遊戲的明確定義，並且以這個定義為基礎，設計一個魔法牌遊戲的開發系統，讓大家可以透過這個開發系統來設計魔法牌遊戲。

1.4 論文章節介紹

我們會在第二章提出魔法牌遊戲的定義與魔法牌遊戲的系統架構，在第三章我們實作魔法牌遊戲開發系統。在第四章透過我們的魔法牌遊戲開發系統實作在 1.2 節所提到的「99 魔法牌」，並且說明如何透過該系統來擴充牌卡與遊戲規則。最後在第五章對本論文作個總結，並提出一些可行的未來發展方向。

第二章、系統架構

本章節以「魔法牌遊戲」為討論的重點，我們將於 2.1 節為魔法牌遊戲做一個明確的定義。在 2.2 節根據定義提出魔法牌遊戲的系統架構，並於 2.3 節提出以我們在 2.2 節所提的魔法牌遊戲的系統架構為目標的「魔法牌遊戲開發系統」的設計。

2.1 魔法牌遊戲之定義

透過蒐集研究許多魔法牌遊戲（諸如：蒼穹霸主 2[15]、仙境傳說 TCG[8]、永恆之夢[16]、魔法風雲會[2][4]、遊戲王[3][5]、神樣的年代記[17]、Astral Master[18]），我們分析其共通點並提出魔法牌遊戲的定義。

一個遊戲要稱之為魔法牌遊戲，需符合以下的特性：

2. 玩家透過使用遊戲專用牌進行遊戲，這些遊戲專用牌上附有遊戲規則敘述，玩家需遵循。
3. 為了能夠調整遊戲的平衡性與增加遊戲的趣味性，遊戲專用牌的種類允許增加與擴充，也就是說遊戲規則可透過增加新牌的方式增加與擴充。
4. 玩家每次從自己的所有牌卡中依某種規則選取一小部份的牌卡進行遊戲。
5. 有明確的樹狀物件架構（圖 1），敘述如下：
   - 一個「遊戲」可擁有 n 位「玩家」。
   - 一位「玩家」可擁有 m 個「牌區」。
   - 一個「牌區」可放置 k 張「牌卡」。
   - 一張「牌卡」可擁有 p 個「遊戲規則敘述」
6. 回合制，明確規定遊戲裡的每個時間點，所有的玩家可以做什麼事情。

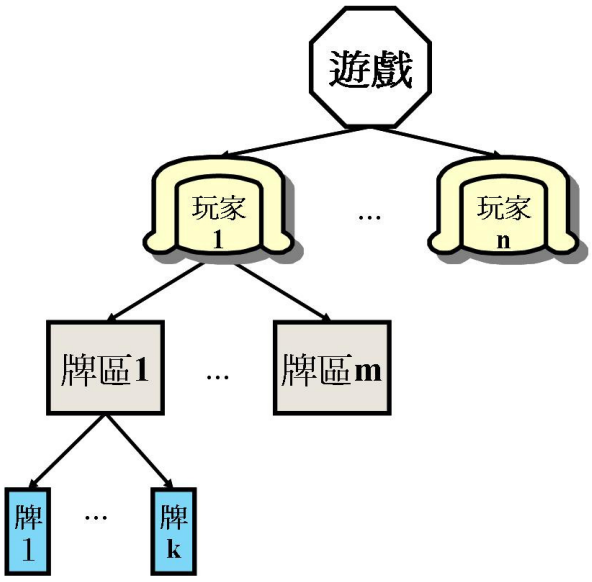


圖 1：魔法牌遊戲樹狀物件架構

2.2 魔法牌遊戲系統架構

根據 2.1 節魔法牌遊戲定義裡的第四項，魔法牌遊戲本身有個明確的樹狀物件架構，所以我們將魔法牌遊戲以物件化的方式來呈現，這部份我們會在 2.2.1 說明。再根據 2.1 節魔法牌遊戲定義裡的第五項，我們設計一個階段系統來做到回合制的需求，這部份我們將於 2.2.2 節說明。最後，因為一款魔法牌遊戲需要設定相當多的遊戲規則，諸如「遊戲名稱」、「玩家人數」、「牌組限制」、「勝敗條件」、「遊戲流程」、「牌區設定」等。因為規則設計繁雜，所以我們需要有個可以設定與管理遊戲規則的系統。這個系統我們稱之為「動作系統」，詳細將於 2.2.3 節說明。

2.2.1 物件化

魔法牌遊戲的物件類別有下列幾種：

- 遊戲－儲存整個遊戲的共用資訊，如回合數、遊戲設定。
- 玩家－參與遊戲的玩家。

- 牌區－放置牌的區域。不同玩家能看到的牌區會有所差異[19]。
- 牌卡－遊戲元件，牌卡上面有規則敘述，玩家透過這些規則敘述來操作遊戲。
- 階段－遊戲流程設計所使用的元件。
- 動作－指令操作的物件。遊戲流程的運作與玩家的操作都是透過該物件來執行，也就是說所有的規則全部都是透過動作物件來設計。

在這些物件當中，「遊戲」、「玩家」、「牌區」、「牌卡」四種物件裡會有遊戲所需要的一些屬性值，這些屬性值提供「動作」物件在進行遊戲操作時的判斷與參考。「階段」則為遊戲流程設計所專用的物件，「動作」物件則是控制整個遊戲所有指令操作的物件，遊戲流程的運作與玩家的操作都是透過「動作」物件來執行。

「遊戲」、「玩家」、「牌區」、「牌卡」這些物件，我們又稱之為「遊戲基本物件」，這些物件皆有因不同遊戲所需要而異的各項屬性值，他們最主要的作用就在於提供動作物件的指令設定與參考。以「99 魔法牌」為例，遊戲物件擁有「整數:遊戲積數」這個屬性值，提供玩家出牌時改變，還有做勝敗判斷所用；玩家物件擁有「玩家:下家」這個屬性值，提供遊戲流程中要決定下一個輪到的玩家時使用；牌卡物件則有「整數:數字」這個物件，提供玩家在出牌做加或是減的操作的時候，改變遊戲的「整數:遊戲積數」這個屬性值的參考的依據。表 2 為「99 魔法牌」的基本物件屬性值一覽表。

類別名稱	屬性名稱
遊戲	整數:遊戲積數、玩家:輪到的玩家
階段	無
玩家	玩家:下家、玩家:上家
牌區	無
牌卡	整數:數字
動作	無

表 2：「99 魔法牌」的物件類別屬性值一覽表

一般而言，牌卡物件上有許多的動作物件，玩家可以透過使用牌卡上的動作物件來進行對遊戲的操作。但不只牌卡物件上可以有動作物件來對遊戲進行操作，其他的物件亦可以有動作物件提供玩家對整個遊戲進行操作。以「99 魔法牌」為例，當玩家要出『５』這張牌時，玩家可以使用『５』這個牌卡物件上的『加』這個動作物件，來將遊戲的「整數:遊戲積數」做加五的動作；當有的玩家不想玩的時候，可以使用玩家物件上的『投降』這個動作物件來宣告認輸；當玩家覺得自己抽牌的運氣太差時，可以使用自己的『牌庫』這個牌區物件上的『洗牌』這個動作物件來洗牌，改變抽牌的順序。

所以，這些遊戲基本物件，一方面可以成為動作物件的執行指令參考與改變的對象，另一方面，他們也可以擁有動作物件，提供玩家來對遊戲進行操作。

這些物件之間的關係如圖 2 所示，說明如下：

- 一個「遊戲」可擁有多位「玩家」。
- 一位「玩家」可擁有多個「牌區」。
- 一個「牌區」可放置多張「牌卡」。
- 「遊戲」、「玩家」、「牌區」、「牌卡」上都可有多個「動作」。

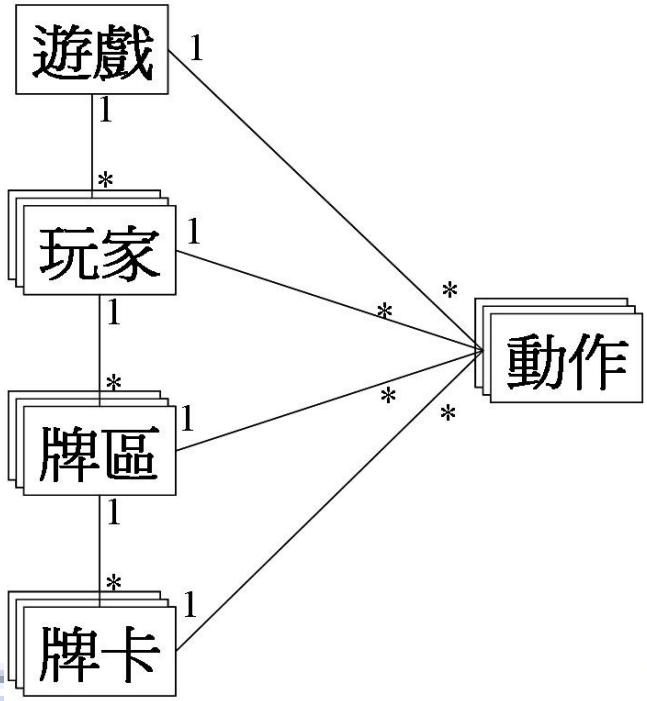


圖 2：魔法牌遊戲物件關係圖

2.2.2 階段系統

根據 2.1 節魔法牌遊戲定義裡的第五項，魔法牌遊戲是個回合制的遊戲，不僅需要一個明確的遊戲流程，還需要在遊戲流程中對所有玩家做明確的行動限制，譬如只在玩家可以出牌的時候才能讓玩家出牌。

我們使用階段系統來達到回合制與明確的遊戲流程的需求。階段物件是個裝著動作物件的容器，一個階段物件可裝著數個動作物件，而遊戲流程則是由數個階段物件所組成。每個階段物件皆有一個指著下一個階段物件的指標，所以遊戲流程裡的階段物件們本身會形成一個循環。

遊戲流程的執行，是從某一個被設為開頭的階段物件開始，依序執行這個階段物件裡的所有動作物件，全部執行完之後，再到下一個

階段物件裡，去依序執行下一個階段物件裡的所有動作物件。如此一直不斷地循環，直到執行到某個讓遊戲結束的動作物件為止。

以「99 魔法牌」為例，我們可以說「99 魔法牌」是由下列三個階段(圖 3)所組成的：

1. 換誰階段－決定現在輪到誰(一般而言是輪到剛才出牌的玩家的下家)

2. 出牌階段－允許現在輪到的玩家可以出一張牌。

3. 抽牌階段－允許現在輪到的玩家可以抽一張牌。

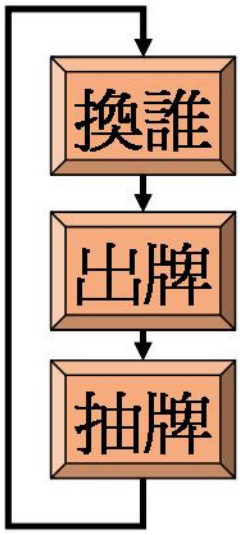


圖 3：「99 魔法牌」的階段流程

這三個階段物件裡都有一些動作物件來執行遊戲流程中的遊戲規則，「99 魔法牌」的遊戲流程就是在這三個階段裡不斷地循環，在每個階段裡依序執行內部的指令（也就是動作物件），直到執行到某個讓遊戲結束的指令。

2.2.3 動作系統

動作系統是本論文最重要的一個設計。我們將在 2.2.3.1 節討論動作系統需要有的特性，在 2.2.3.2 節裡討論動作系統的設計。

2.2.3.1 動作系統的特性

為了統一所有規則的設計，我們將魔法牌的所有元件全部都物件化，並且將系統流程與玩家操作的所有指令也用物件化的方式，將之

以動作物件來包裝，透過執行動作物件的方式來運作整個遊戲。這個透過動作物件來操作整個遊戲的動作系統需要具備一般性與擴充性這兩項特性。

一般性

因為魔法牌遊戲的運作完全透過動作系統，所以動作系統必需具備一般性，使得遊戲可以透過動作系統做到一切的規則需求。為了達到一般性，動作系統至少支援以下的功能：

- 複合式動作(含變數指定、函式呼叫) (compound actions)
- 條件判斷 (if-else)
- 迴圈 (for loop , while loop)
- 輸入輸出控制 (I/O handling)

當動作系統支援以上這些功能的時候，動作系統本身就具備了和程式語言相同的強度，也就是說動作系統其實是一種經過包裝過後的程式語言，透過動作系統來操作遊戲，其實就是經由動作系統去執行程式來操作遊戲。

擴充性

在 2.1 節魔法牌遊戲定義裡的第二項，我們提到「為了能夠調整遊戲的平衡性與增加遊戲的趣味性，魔法牌遊戲需要允許增加新的牌卡與新的遊戲規則」。因為在魔法牌遊戲裡，所有的玩家操作都是透過玩家所帶的牌，依照牌上的規則敘述來執行，所以，其實牌上的規則敘述就是玩家可以操作的動作(如表 3)，而當遊戲公司發行新的牌卡的時候，就可以將一些新的規則敘述寫在新的牌卡上，如此，就能透過發行新牌卡的方式，來增加新的遊戲規則了。

牌	遊戲規則敘述	動作
A	遊戲積數+1	『加』
2	遊戲積數+2	『加』
3	遊戲積數+3	『加』
4	遊戲積數+4	『加』
5	遊戲積數+5	『加』
6	遊戲積數+6	『加』
7	遊戲積數+7	『加』
8	遊戲積數+8	『加』
9	遊戲積數+9	『加』
10	遊戲積數+10 or 遊戲積數-10	『加』、『減』
J	Pass	『pass』
Q	遊戲積數+20 or 遊戲積數-20	『加』、『減』
K	遊戲積=99	『99』
Joker1	別人用減之後強迫他從手上隨機棄掉一張牌	『丟牌』
Joker2	別人用減之前遊戲積數先減 5，再強迫他出一張牌	『再出』

表 3：「99 魔法牌」各種牌卡的動作一覽表

但是，在新增遊戲規則的時候，很實際的一點是，因為已經發售了的牌卡我們無法改變其上的遊戲規則敘述，所以，當我們想要透過新增遊戲規則的方式來改變舊牌上的遊戲規則的時候，我們就必需透過間接的方式，「讓新牌的遊戲規則伴隨在舊牌的遊戲規則的前後」來對舊牌的遊戲規則做改變。

譬如在「99 魔法牌」裡，考量到『10』和『Q』這兩張牌的『減』的這個動作太氾濫(如圖 4(a) , (b))，希望可以增加玩家在使用『減』這個動作的時候的風險，於是我們就設計了一張新的牌叫『Joker1』(圖 4(c))，並給予這張牌的一個伴隨在『減』這個動作之後的動作，叫做『丟牌』，效果是：這個動作會在別人使用『減』這個動作之後

被執行到，然後讓剛才使用『減』的這個動作的玩家從手上隨機棄掉一張牌。

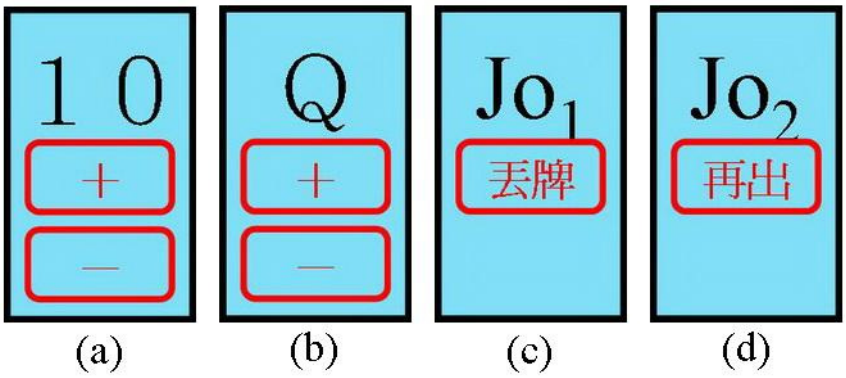


圖 4：「99 魔法牌」裡的 10、Q、Joker

例如：玩家 1 的手上有『Joker1』這張牌，然後這個時候換玩家 2 出牌，玩家 2 出了『10』這張牌，使用了『10』這張牌上的『減』這個動作時，此時『減』這個動作在執行之後會自動去觸發玩家 1 手上的『Joker1』這張牌的『丟牌』這個動作，讓玩家 2 在『減』之後，從手上隨機棄掉一張牌。

再譬如，我們可以設計另一張新牌叫『Joker2』(圖 4 (d))，並給予這張牌的一個伴隨在『減』這個動作之前的動作，叫做『再出』，效果是：這個動作會在別人使用『減』這個動作之前，先把「遊戲」的「遊戲積數」減五，然後要求剛才用『減』的那個人再出一張牌，如果此時剛才使用『減』的玩家手上沒有其他可以使用的牌，那他就輸了。

例如：玩家 1 的手上有『Joker2』這張牌，然後這個時候換玩家 2 出牌，且此時的遊戲積數=99，當玩家 2 出了『10』這張牌，使用了『10』這張牌上的『減』這個動作時，此時『減』這個動作在執行之前會自動去觸發玩家 1 手上的『Joker2』這張牌的『再出』這個動作，先將遊戲積數-5(此時遊戲積數=94)，再強迫玩家 2 出一張牌，如果此時玩家 2 手上沒有可以出的牌，那玩家 2 就輸了！

透過這兩個例子我們可以發現，這些加入的新規則能夠適時地增

加『減』的風險，進而達到在不更改舊牌的遊戲規則的前提之下，做到更改舊牌的遊戲規則的效果。

透過這種「讓新牌的遊戲規則伴隨在舊牌的遊戲規則的前後」的機制，我們可以成功地在不改變舊牌的遊戲規則敘述的前提下，用新牌的規則來改變舊牌的規則。而這種伴隨在別的規則前後的規則，我們稱之為觸發式規則，而在動作系統中，這種伴隨在別的動作物件前後的動作物件，我們就稱之為觸發式動作。

2.2.3.2 動作系統的設計

在這一節裡，我們將針對動作系統的一切需要做適當的設計。

觸發式動作

「觸發式動作」指的就是這個動作物件會被別的動作物件所觸發，譬如在前面的例子裡，『Joker1』這張牌上的『丟牌』這個「動作」物件會被『10』或是『Q』的『減』這個動作物件在執行之後觸發，而『Joker2』這張牌上的『再出』這個動作物件會被『10』或是『Q』的『減』這個動作物件在執行之前觸發。

為了能夠達到觸發式動作的設計，我們必需在執行每個動作物件的前與後檢查是否有其他動作物件需要被執行。於是，我們讓每個動作物件都擁有一個「前置動作佇列」與「後置動作佇列」，其定義如下：

- 前置動作佇列(preAction List)：
  儲存所有在該動作物件執行之前可能會被執行的動作。在執行該動作物件之前，先執行該佇列裡的所有動作物件。

- 後置動作佇列(postAction List)：
  儲存所有在該動作物件執行之後可能會被執行的動作。在執行該動作物件之後，再執行該佇列裡的所有動作物件。

也就是說，一個動作物件本身的執行程序應該是：

1. 執行該動作的所有前置動作。
2. 執行該動作的執行效果。
3. 執行該動作的所有後置動作。

在前面的例子裡，我們可以說『再出』這個動作是『減』的前置動作，而『丟牌』這個動作是『減』的後置動作，所以當要執行『減』這個動作的時候，程序就會變成：

1. 執行『減』的所有前置動作 ➔ 執行『再出』
2. 執行『減』的執行效果 ➔ 讓遊戲積數變小
3. 執行『減』的所有後置動作 ➔ 執行『丟牌』

執行時機(Specification)

經過前面的討論，我們可以知道一個動作物件有可能在下列三種情形中被執行：

1. 在遊戲流程裡被系統呼叫執行
2. 在某個階段裡經由玩家的操作而執行
3. 經由別的動作物件觸發而被執行

所以每個動作物件都需要有一個執行的執行時機來設定這些動作物件的正確執行時機。譬如在「99 魔法牌」的例子裡，『５』這張牌上的『加』的執行時機就是「玩家在出牌階段時可執行」，而『Joker1』這張牌上的『丟牌』的執行時機就是「在『減』這個動作之後觸發執行」。

執行條件(Condition)

動作物件除了需要有「執行時機」之外，尚需要有「執行條件」的設定，來檢查一個動作物件在符合可以被執行的時機時，是否真的可以被執行。

譬如在「99 魔法牌」的例子裡，『５』這張牌上的『加』的執行時機雖然是「玩家於出牌階段時可執行」，但『加』這個動作仍然需要加上一些執行條件的限制，例如「這張牌的擁有者是現在輪到的玩家」、「遊戲積數加上這張牌的數字不會超過 99」，如此才不會在每個玩家手上都持有『５』這張牌的時候，就可以任意地使用『加』這個動作。

又譬如在「99 魔法牌」的例子裡，『Joker1』這張牌的『丟牌』動作會在有人使用『減』動作時被執行，但仍要加上一項執行條件限制：「使用『減』的玩家與這張牌的擁有者不同」否則當有玩家在自己手上有『Joker1』時使用別的牌的『減』的動作，結果會因為『減』而觸發到自己的『Joker1』的『丟牌』動作而逼自己還要多丟掉一張牌，這就與我們的設計原意大不相同了。

執行效果(Operation)

當一個動作物件在通過執行條件的檢查，確定可以被執行時，會去執行的指令。

為了達到一般性的需求，執行效果需要支援至少以下的功能：

1. 複合式動作(含變數指定、函式呼叫) (compound actions)
2. 條件判斷 (if-else)
3. 迴圈 (for loop , while loop)
4. 輸入輸出控制 (I/O handling)

並且為了能夠達到擴充性的需求，所以執行效果的內容應該是從玩家所帶的牌組裡讀入之後才去做動態的設定，而非一開始先把所有的執行效果指令全部寫死在遊戲裡。所以執行效果的部份，我們採用讀取指令集的方式，先定義好數種能夠支援一般性需求的指令集與參數格式之後，再針對每種指令集做動態的設定就可以了。

2.3 魔法牌遊戲開發系統設計

我們希望可以設計一個開發系統來專門開發魔法牌遊戲，經由 2.1 節與 2.2 節的分析，我們知道，這個系統在設計一個魔法牌遊戲時，需要能做到以下幾點：

1. 可以讓使用者設定樹狀物件架構(各個物件的個數)
2. 可以讓使用者自行在各個類別上新增遊戲所需要的各種屬性
3. 可以讓使用者設定每個物件的各種屬性值
4. 可以動態產生所有的物件

第三章、魔法牌遊戲系統實作

在 3.1，我們會為整個系統做個概要說明，然後於 3.2 節說明「遊戲基本物件」(「遊戲」、「玩家」、「牌區」、「牌卡」) 的設計。在 3.3 節提到「階段系統」的設計，在 3.4 節提到「動作系統」的設計。最後在 3.5 節提到其他底層的一些重要設計。

3.1 系統概要

根據 2.1 節魔法牌遊戲之定義中的第四點所提到的「樹狀物件架構」，我們允許「遊戲」、「玩家」、「牌區」、「牌卡」擁有「動作」物件提供玩家對遊戲做操作。「階段」物件亦擁有「動作」物件，並於遊戲中透過一個不斷循環的迴圈，依序執行每個「階段」物件裡的每個「動作」物件。「動作」物件透過「執行時機」將「動作」物件做允許觸發的時間處理(譬如將某個觸發動作放置於可以觸發該動作的前置動作佇列或後置動作佇列裡)。至於「執行條件」與「執行效果」則全透過指令集的方式來動態處理。

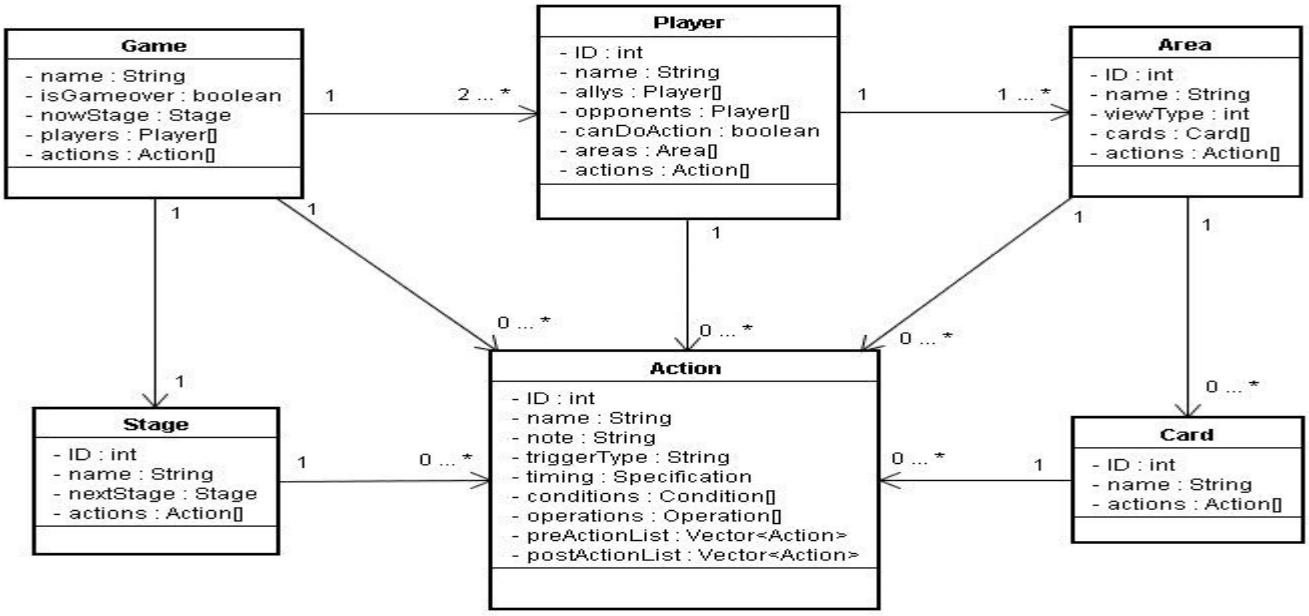


圖 5：魔法牌主要物件的 class diagram

3.2 遊戲基本物件

遊戲基本物件指的是「遊戲」、「玩家」、「牌區」、「牌卡」這四種物件，這四種物件組成樹狀物件架構，如圖 2 所示。這些物件的 class diagram 如圖 6 所示。在此我們為各個類別做詳細的介紹：

遊戲(Game)

屬性值名稱	型態	說明
name	String	遊戲名稱。
isGameover	Boolean	遊戲是否結束的變數。詳細請見 3.1.3。
nowStage	Stage	現在所在的階段。詳細請見 3.1.3。
players	Player[]	這個遊戲的所有玩家。
actions	Action[]	這個遊戲所擁有的「動作」物件操作。

表 4：遊戲(Game)類別的詳細說明

一個魔法牌遊戲只有一個「遊戲」物件，這個遊戲物件代表的就是整個遊戲，故這個遊戲會有一個指向目前所在的階段的 nowStage 指標，然後由這個 nowStage 開始執行不斷循環的遊戲流程。關於遊戲流程的運作，請見 3.1.3。

「遊戲」物件在程式建置、遊戲玩家尚未到齊、遊戲尚未開始的時候就已經存在。

玩家(Player)

屬性值名稱	型態	說明
ID	Int	玩家在遊戲裡的唯一編號。
name	String	玩家於遊戲裡的名稱。
allys	Player[]	同盟的玩家們。在設定牌區的視野的時候會用到。
opponents	Player[]	敵隊的玩家們。在設定牌區的視野的時候會用到。
canDoAction	boolean	設定這個玩家現在是否可以執行指令。在遊戲規則設定中，如果要讓玩家可以使用「動作」物件來操作遊戲時，需要先將該玩家的這個變數設為 true 才行。
areas	Area[]	這個玩家所擁有的牌區。
actions	Action[]	這個玩家所擁有的「動作」物件操作。

表 5：玩家(Player)類別的詳細說明

一個魔法牌遊戲只有等同於玩家個數的「玩家」物件。

「玩家」物件在程式建置、遊戲玩家尚未到齊、遊戲尚未開始的時候就已經存在。但「玩家」物件裡的 ID 與 name 這兩項屬性值需要在遊戲玩家加入時，從外部讀取。

牌區(Area)

屬性值名稱	型態	說明
ID	Int	這個牌區的唯一編號。
name	String	這個牌區的名稱。
viewType	int	這個牌區可以被檢視的型態。在這裡我們採用 bitwise 的操作。 bit 1： 自己可以檢視。 bit 2： 同盟可以檢視。 bit 3： 敵隊可以檢視。 例：7 = 0111 ➔ 所有人都可以檢視 3 = 0011 ➔ 自己和同盟可以檢視
cards	Card[]	這個牌區上面的牌。
actions	Action[]	這個牌區所擁有的「動作」物件操作。

表 6：牌區(Area)類別的詳細說明

一個魔法牌遊戲共有「玩家數 x 每位玩家所擁有的牌區數」個數的「牌區」物件。

「牌區」物件在程式建置、遊戲玩家尚未到齊、遊戲尚未開始的時候就已經存在。至於每個牌區裡有哪些「牌卡」物件，則要在遊戲玩家加入時讀入他們的牌組資訊，然後才動態設定。

牌卡(Card)

名稱	型態	說明
ID	Int	這張牌卡的唯一編號。
name	String	這張牌卡的名稱。
actions	Action[]	這張牌卡所擁有的「動作」物件操作。

表 7：牌卡(Card)類別的詳細說明

一個魔法牌遊戲共有「所有玩家的牌組裡的牌卡數」個數的「牌卡」物件。

「牌卡」物件在程式建置、遊戲玩家尚未到齊、遊戲尚未開始的時候尚未存在。這些物件需要在遊戲玩家加入時讀入他們的牌組資訊之後才動態產生。

3.3 階段流程

階段的類別說明如下：

階段(Stage)

名稱	型態	說明
ID	Int	這個階段的唯一編號。
name	String	這個階段的名稱。
nextStage	Stage	這個階段的下一個階段。
actions	Action[]	這個階段裡要執行的所有動作。

表 8：階段(Stage)類別的詳細說明

階段流程的部份，我們使用一個 while 迴圈在各個階段中不斷循環，直到遊戲結束。每個階段都依序執行該階段裡的所有動作物件。在這裡，會使用到遊戲物件裡的 isGameover 與 nowStage 屬性值。相關的程序如下：

```
Game.start() {
   While(isGameover) {
      nowStage.doThisStage() ;
      nowStage = nowStage.nextStage ;
   }
}
```

```
Stage.doThisStage() {
   for(int f1 = 0; f1<actions.size(); f1++) {
      execAction(actions.get(f1) ;
   }
}
```

「階段」物件在程式建置、遊戲玩家尚未到齊、遊戲尚未開始的時候就已經存在。

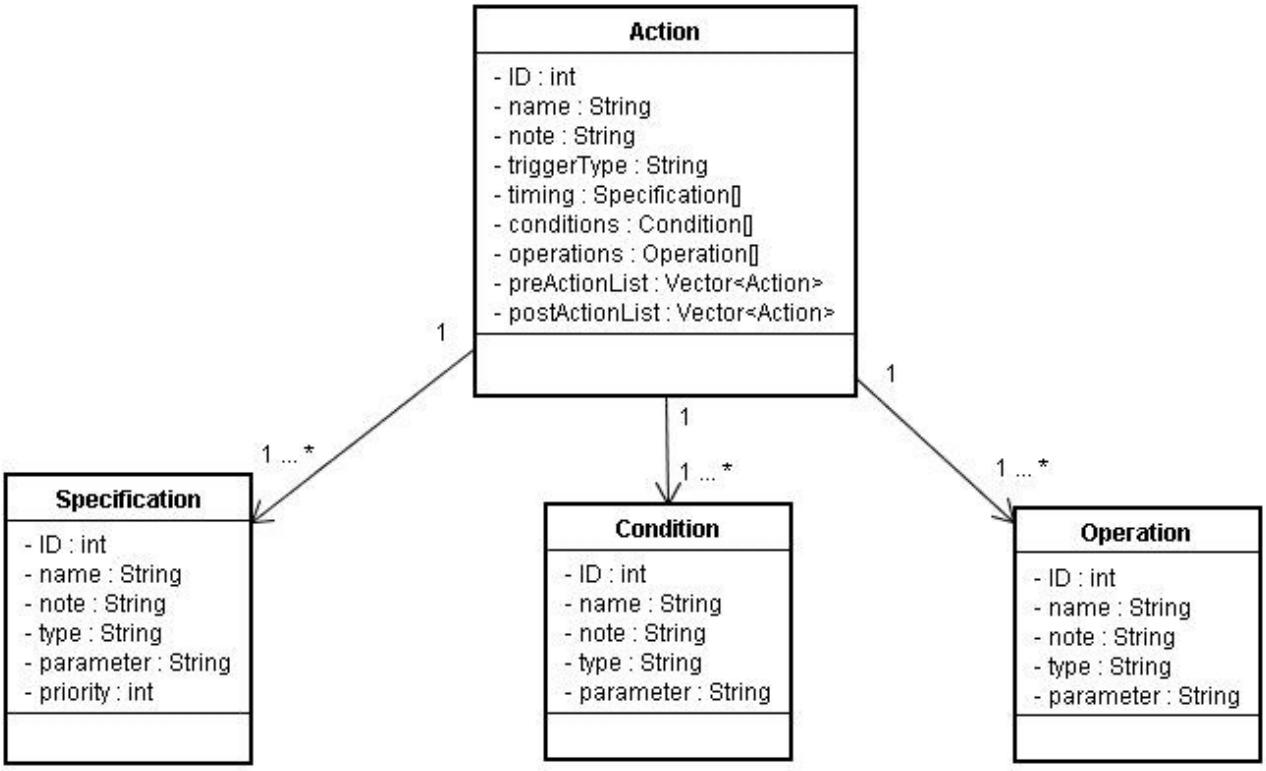


圖 6：動作系統相關設定的 class diagram

3.4 動作系統

動作系統是這個魔法牌遊戲開發系統的核心，在 2.2.3 節，我們介紹了動作系統的特性與設計，並且提到三項設計：執行時機(Specification)、執行條件(Condition)、執行效果(Operation)，我們將這些相關的設計的 class diagram 表示於圖 6，並在這裡為每個 class 做介紹：

動作(Action)

名稱	型態	說明
ID	Int	這個動作的唯一編號。
Name	String	這個動作的名稱。
Note	String	這個動作的註解。
triggerType	String	這個動作的觸發型態，這會在設定執行時機的時候用到。
Timings	Specification[]	這個動作的執行時機。
Conditions	Condition[]	這個動作的執行條件。
Effects	Operation[]	這個動作的執行效果。
preActionList	Vector<Action>	這個動作的前置動作佇列。
postActionList	Vector<Action>	這個動作的後置動作佇列。

表 9：動作(Action)類別的詳細說明

我們可以把動作看成是程式語言裡的一個函式，執行時機是指「這個動作(函式)會被呼叫的時機」；執行條件是指這個動作(函式)被呼叫時，是否會被執行。如果條件成立那就會執行，反之不會；執行效果是當這個動作(函式)被呼叫，而且條件亦成立的時候，會做的操作。而前置動作佇列與後置動作佇列則存著一些會被這個動作所觸發的其他動作。當這個動作(函式)被呼叫且條件亦成立時，會在執行效果前先執行前置動作佇列裡的所有動作，然後再執行這個動作的執

行效果，接著再執行後置動作佇列裡的所有動作。一個動作物件的執行程序如下面這段程式碼所述：

因為「動作」物件一部份來自階段流程裡的階段，一部份來自遊戲基本物件，而遊戲基本物件裡的「牌卡」物件又是在遊戲玩家將所使用的牌組資料帶入遊戲時，讀入牌組資料時才產生的，所以「動作」物件在程式建置、遊戲玩家尚未到齊、遊戲尚未開始的時候尚未存在。這些物件需要在遊戲玩家加入時讀入他們的牌組資訊之後才動態產生。

執行時機(Specification)

名稱	型態	說明
ID	Int	這個執行時機的唯一編號。
name	String	這個執行時機的名稱。
note	String	這個執行時機的註解。
type	String	這個執行時機的類別。
parameter	String	這個執行時機的參數。
priority	int	這個執行時機的優先序。若有兩個以上的動作的執行時機相同，同被某個動作所觸發，則優先序越小的越先執行，相同的話則隨機。

表 10：執行時機(Specification)類別的詳細說明

動作的執行時機有三種：

1. 階段流程直接使用
2. 玩家在某個階段可以使用
3. 由某個動作在執行前或執行後觸發

所以，我們將這些類別做分類，然後使用 type 與 parameter 這兩項

屬性值來做設定，詳細的設定方式如表 11：

說明	type	parameter
階段流程直接使用	系統呼叫	無
玩家於某階段可以使用	玩家使用	階段名稱
某種動作的前置動作	動作開始前觸發	觸發的動作類別
某種動作的後置動作	動作開始後觸發	觸發的動作類別

表 11：執行時機設定一覽表

如果我們要設定一個動作是某個動作的前置動作或後置動作，譬如在「99 魔法牌」裡，我們想要設定『再出』這個動作是『減』這個動作的後置動作，那我們就需要在『再出』這個動作的執行時機裡，type 的地方設為「動作開始後觸發」，然後在 parameter 裡，填入『減』這個動作的觸發類別，也就是『減』這個動作物件的 triggerType 這個屬性值的值。

與「動作」物件相同，「執行時機」物件在程式建置、遊戲玩家尚未到齊、遊戲尚未開始的時候尚未存在。這些物件需要在遊戲玩家加入時讀入他們的牌組資訊之後才動態產生。

執行條件(Condition)

名稱	型態	說明
ID	Int	這個執行條件的唯一編號。
name	String	這個執行條件的名稱。
note	String	這個執行條件的註解。
type	String	這個執行條件的類別。
parameter	String	這個執行條件的參數。

表 12：執行條件(Condition)類別的詳細說明

執行條件是在一個動作物件要被執行時需要先做的判斷，所以需要參考一個布林值，或是一個布林運算式，在此，我們仍然使用 type 與 parameter 這兩個屬性值來做設定，詳細的設定方式如表 13：

說明	type	Parameter
階段流程直接使用	Boolean	布林值
玩家於某階段可以使用	Expression	布林運算式

表 13：執行條件設定一覽表

因為布林運算式需要用到遊戲裡的物件的屬性值做參考，而遊戲裡的物件類別可以讓遊戲設計師自行設計屬性值，所以在這裡需要可以動態判斷「執行條件」物件的條件設定是否符合。我們使用與「執行效果」相同的讀取指令集的方式來處理。我們設計一套文法(grammar)，然後讓「執行條件」的參數(parameter)需要能被該文法所辨識，如此我們就可以針對該文法所產生的所有執行條件格式一一做處理。關於文法的部份，詳見附錄 A。

與「動作」物件相同，「執行條件」物件在程式建置、遊戲玩家尚未到齊、遊戲尚未開始的時候尚未存在。這些物件需要在遊戲玩家加入時讀入他們的牌組資訊之後才動態產生。

執行效果(Operation)

名稱	型態	說明
ID	Int	這個執行效果的唯一編號。
name	String	這個執行效果的名稱。
note	String	這個執行效果的註解。
type	String	這個執行效果的類別。
parameter	String	這個執行效果的參數。

表 14：執行效果(Operation)類別的詳細說明

執行效果是在一個動作符合執行條件之後要去對整個遊戲做的操作。所以需要能夠符合 2.2.3.1 節裡所提的一般性。我們仍使用 type 與 parameter 來做設定。詳細的設定方式如表 15 所示。

一般性需求	指令集類別	參數
複合式動作	綜合效果	**[效果]** AND **[效果]** / **[效果變數]**
	變數指定	**[變數 A] [指定運算子] [變數 B]**
	函式呼叫	**<選擇函式>**
條件判斷	If	if **[條件]** then **[效果]**
	If-else	if **[條件]** then **[效果]** else **[效果]**
迴圈	For	FOR **[整數]** = **[整數]** To **[整數]** do **[效果]**
	While	WHILE(**[條件]**) do **[效果]**
輸入輸出控制	控制權	系統 / 玩家
	顯示文字	顯示 **[字串]**

表 15：執行效果設定一覽表

與執行條件一樣，因為執行效果所要改變的對象是遊戲裡的物件，而這些物件所擁有的屬性值會因遊戲設計師的設計而有所不同，所以我們仍然透過我們在「執行條件」的部份所提的文法(grammar)

來設定，「執行效果」的參數(parameter)需要能被該文法所辨識，如此我們就可以針對該文法所產生的所有執行效果格式一一做處理。關於文法的部份，詳見附錄 A。

關於文法的部份，因為執行條件與執行效果都需要透過文法來產生符合正確格式的設定參數，而這些設定參數又有可能會指向玩家所設定的類別屬性值，譬如「牌卡.數字」這就是「99 魔法牌」這個遊戲所需要用到的一個自定的屬性值，所以文法的部份亦需要在遊戲設計師設定好自定的物件類別屬性值之後，才動態產生。

3.5 其他底層的重要設計

為了讓「執行效果」與「執行條件」能夠存取到整個樹狀物件架構的所有物件，所有的物件類別（遊戲、階段、玩家、牌區、牌卡、動作）都有「擁有者指標」，透過這些指標，可以讓每個物件取得自己的擁有者：

- ownerGame
- ownerStage
- ownerPlayer
- ownerArea
- ownerCard
- ownerAction

另外，因為有「觸發式動作」的設計，所以動作物件亦有 triggerAction 這一個指向觸發自己的動作的指標。

3.6 魔法牌遊戲開發系統之設計

因為本系統需要能提供遊戲設計者設計各式各樣的魔法牌遊戲所以在上述所提到的每種物件，都需允許遊戲設計者自行定義遊戲所需要的各項屬性值，譬如在「99 魔法牌」裡，牌卡的屬性值只有「數字」而已，但在別的遊戲裡，如「魔法風雲會」，牌卡的屬性值可能就還需要「攻擊力」與「防禦力」等等其他的屬性值。故本系統需要允許遊戲設計者在各種物件上自行定義遊戲所需要的屬性值。

因為魔法牌遊戲的牌與規則是可以增加的，所以在牌卡的設定上，我們採用讀取牌卡設定檔的方式，如此當我們要增加新牌卡的時候只要修改牌卡設定檔即可。

在設定完畢之後，會得到一個遊戲設定組態檔與這個遊戲的所有程式碼，這些程式碼主要由三個部份組成：

1. 根據使用者定義的其他擴充屬性值而有的變數存取相關程式碼
2. 該物件在系統裡內建要有的功能程式碼
3. 由使用者自定的擴充函式程式碼（詳見附錄Ｃ）

當使用者透過該系統做完設計後，編譯該系統所產生的程式檔，這些程式檔便會去讀取由使用者設計出來的設定組態檔，然後建置遊戲，並開始執行遊戲。關於遊戲的執行流程，詳見附錄Ｃ。

第四章、使用本系統開發魔法牌遊戲

在這一章裡，我們將用我們所開發的魔法牌遊戲開發系統實作「99 魔法牌」。4.1 節講解我們所使用的魔法牌遊戲開發系統的系統環境簡介。4.2 節裡，我們將說明設計一款魔法牌遊戲的程序，4.3~4.6 節實作「99 魔法牌」。並於 4.7 節裡提到擴充魔法牌遊戲的方式。

4.1 系統環境簡介

為了可以動態產生物件，所以我們使用 Java[20]來實作系統。我們使用 JDK1.5，並且提供 GUI 介面讓遊戲設計師可以方便設計遊戲。

本系統對於每一款遊戲皆會產生該遊戲的設定組態檔、java 檔與一組 client-server 的網路架構，設計出來的遊戲邏輯程式放置於 client 端。client 端負責將玩家所使用的牌組傳送給 server，而 server 端則負責接收玩家的連線與玩家所使用的牌組，並於「人數湊齊」時，依照所有玩家傳入的牌組，在遊戲裡動態地產生相對應的牌卡實體物件，然後就能開始遊戲了。

為了測試方便，我們允許 client 端自行持有「牌組」的資料，故 client 端在與 server 端建立連線之後，便可直接傳送牌組資料，而不需要連線至遊戲相關的玩家資料庫下載。

另外，我們也根據 2.1 節魔法牌遊戲之定義中的第四點所提到的「樹狀物件架構」設計了一個遊戲操作介面。該遊戲操作介面只提供對「樹狀物件架構」的呈現與對這些物件裡的動作物件做操作，若遊戲設計師所設計的遊戲尚需其他操作介面，則亦需要自行實作。

4.2 魔法牌遊戲開發程序

根據 2.2 節魔法牌遊戲系統架構我們可以知道，魔法牌遊戲的三個大組成元件分別是：

1. 物件化架構
2. 階段流程
3. 動作系統

我們所提出的魔法牌遊戲開發系統，將透過下列四個程序來讓使用者製作一款魔法牌遊戲：

1. 遊戲基本設定
2. 物件化類別定義
3. 階段流程設定
4. 物件與動作設定

4.3 遊戲基本設定

雖然不同魔法牌遊戲之間的規則差異很大，但仍然有一些相同的部份是可以經由一個共通的設定程序來完成。在這裡的設定，最主要的就是針對遊戲的樹狀物件架構做設定。我們以「99 魔法牌」為範例。設定項目如下(圖 7)：

- 遊戲名稱：遊戲的名稱。譬如「99 魔法牌」。
- 玩家人數：一個遊戲由多少玩家組成。譬如 2 人。
- 牌區數量：每個玩家擁有幾個牌區。譬如 3 個。
- 牌區名稱：設定每個牌區的名稱。譬如「牌庫」、「手上」、「廢牌」。
- 牌區可視限制：設定每個牌區的可視限制。譬如「不能檢視」、「自己可以檢視」、「所有人可以檢視」。
- 牌區起始牌數：設定遊戲一開始時每個玩家的牌在不同牌區的張數。譬如「其他」、「5」、「0」。
- 牌組限制：玩家帶入遊戲的牌組有什麼限制。譬如「張數總數 40 張」、「相同數字的牌最多 4 張」、「特殊牌上限 16 張」

圖 7 ：遊戲基本設定畫面

有了以上的項目，就可以設定好整個遊戲的樹狀物件架構了。整個樹狀架構，應該根據遊戲設計師在這裡的設定而產生，也就是說，在「99 魔法牌」的例子裡，當這些項目如上述設定時，就會自動產生一個遊戲物件，兩個玩家物件，六個牌區物件(每個玩家有三個牌區)。至於牌卡物件的產生，應該於玩家帶牌組進入遊戲時產生。

另外，關於「牌組限制」裡提到的「特殊牌上限 16 張」這項設定，需要明確定義出依據這個遊戲裡牌卡類別的哪個屬性值而決定是否為特殊牌，所以這部份在 4.4 物件化類別定義裡也會有所提到。

4.4 物件化類別定義

設定好了遊戲的樹狀物件架構之後，接著要針對遊戲裡的物件類

別做設定(圖 8)。就像是在寫程式時定義 class 一樣，遊戲設計師需要根據遊戲的需要給予每一種物件類別額外的屬性值。譬如「99 魔法牌」裡，需要一個所有玩家都共用的遊戲積數，所以我們可以設定這個遊戲積數是『遊戲』這種物件的一個屬性值。每個牌卡上面都會有一些數字，所以我們可以設定牌卡類別有一個「整數:數字」這樣子的屬性值。「99 魔法牌」的各個物件類別的屬性值設計，詳見表 2。

另外，如果在遊戲基本設定裡，需要對牌組做一些特別限制的話，亦需要在這裡幫牌卡類別設計一些提供牌組限制用的屬性值。譬如「布林:特殊牌」。

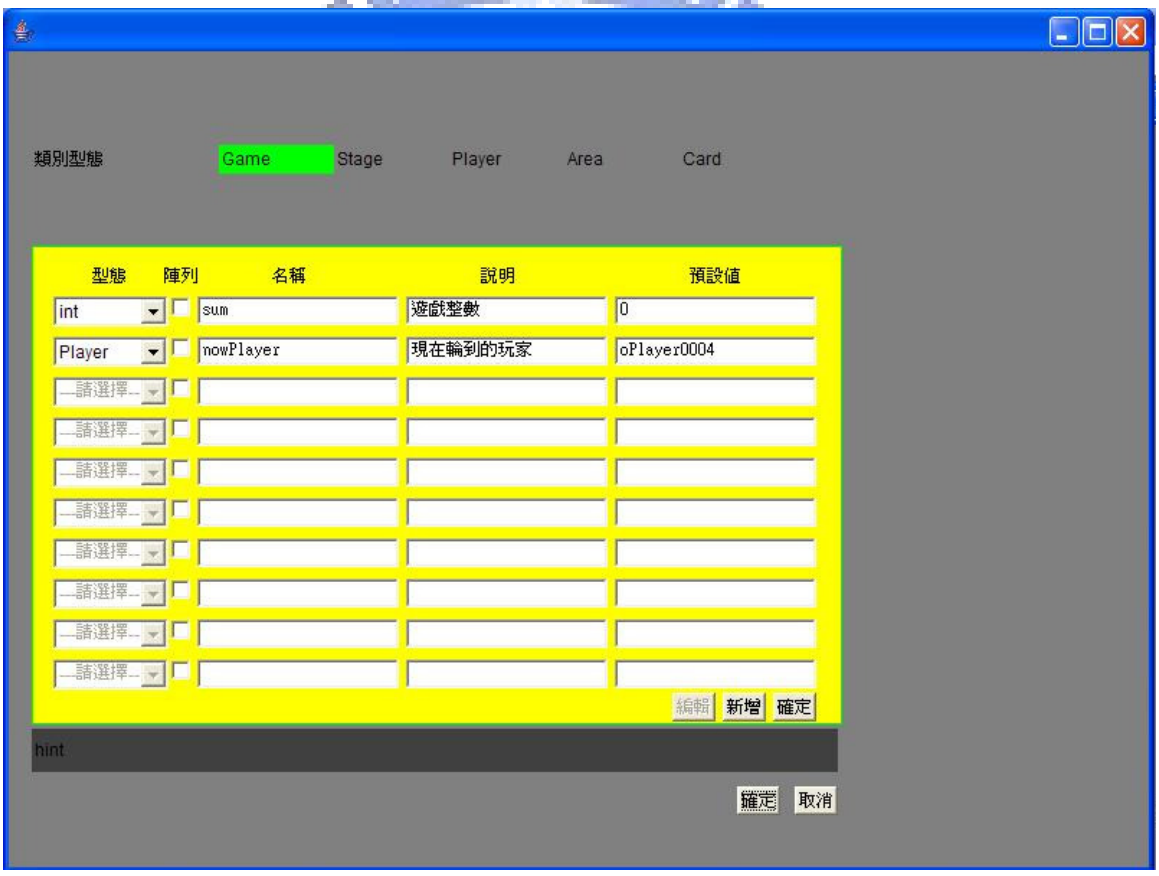


圖 8：物件化類別設定畫面

4.5 階段流程設定

在 2.2.2 節階段系統裡提到，整個魔法牌遊戲的運作是透過一串

形成迴圈的階段物件，依序不斷地去執行各個階段裡的動作物件。所以遊戲設計師需要設定在一款魔法牌遊戲裡，有哪些階段、每個階段叫什麼名字、每個階段會執行哪些動作(圖 9)。在「99 魔法牌」裡，如圖 2-3 所示，一共設定了三個階段：「換誰階段」、「出牌階段」、「抽牌階段」。這些階段的主要要執行的動作如下：

1. 換誰階段
    - 顯示現在的階段名稱。
    - 決定現在輪到誰（一般而言是輪到剛才出牌的玩家的下家）。

2. 出牌階段
    - 顯示現在的階段名稱。
    - 檢查現在輪到的玩家是否輸了(沒牌出)。
    - 顯示現在的遊戲積數。
    - 讓現在輪到的玩家可以出牌。
    - 交出控制權給現在輪到的玩家。

3. 抽牌階段
    - 顯示現在的階段名稱。
    - 讓現在輪到的玩家抽一張牌。

至於這些動作的詳細設定，詳見附錄 B。

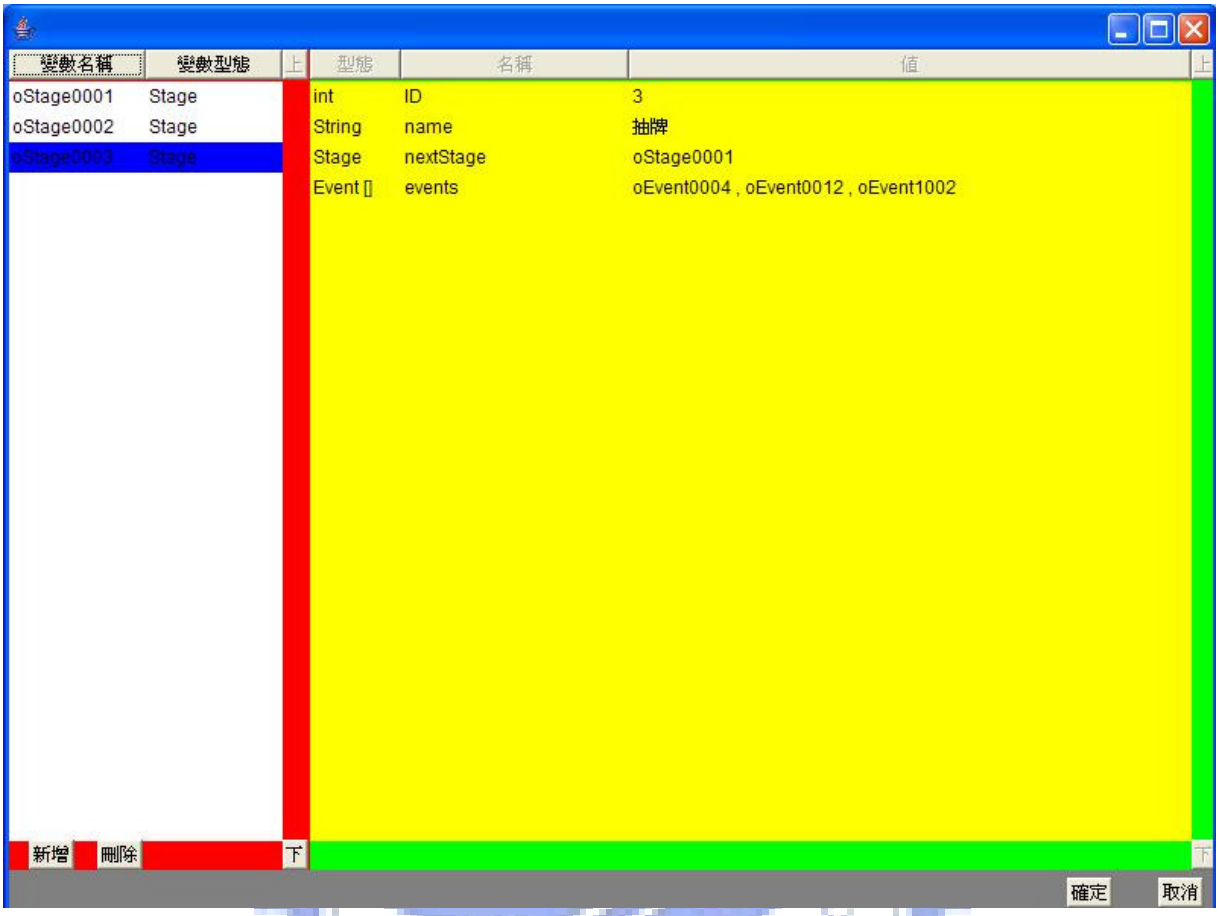


圖 9：階段設定畫面

4.6 物件與動作設定

做完了前三項設定之後，一款魔法牌遊戲的「遊戲」、「階段」、「玩家」、「牌區」這些物件的個數就已經確定了，接著要做的事情是：

1. 設定這些物件的屬性值
2. 設計牌卡物件與動作物件

遊戲設計師需要針對所有的物件做明確的設定，譬如需要對「遊戲」物件的「遊戲積數」變數做設定，使其等於 0；對兩個「玩家」變數裡的「上家」與「下家」做設定，使他們互設為對方。

除了「遊戲」、「階段」、「玩家」、「牌區」這四種物件的個數是經由前面的設定而確定的之外，遊戲設計師可以在這裡自行去設計牌卡物件與動作物件。譬如在「99 魔法牌」裡，遊戲設計師就需要在這裡設計 1~10、J、Q、K、Joker 這些牌，並且去設定這些牌裡的一些屬性值。然後，遊戲設計師還需要去設計整個遊戲所需要的所有的「動作」物件，譬如「99 魔法牌」裡的『加』、『減』、『Pass』、『99』、『丟牌』、『再出』(表 3)，然後將這些動作物件設到擁有這些動作物件的物件上。設定的畫面如圖 10、11 所示。

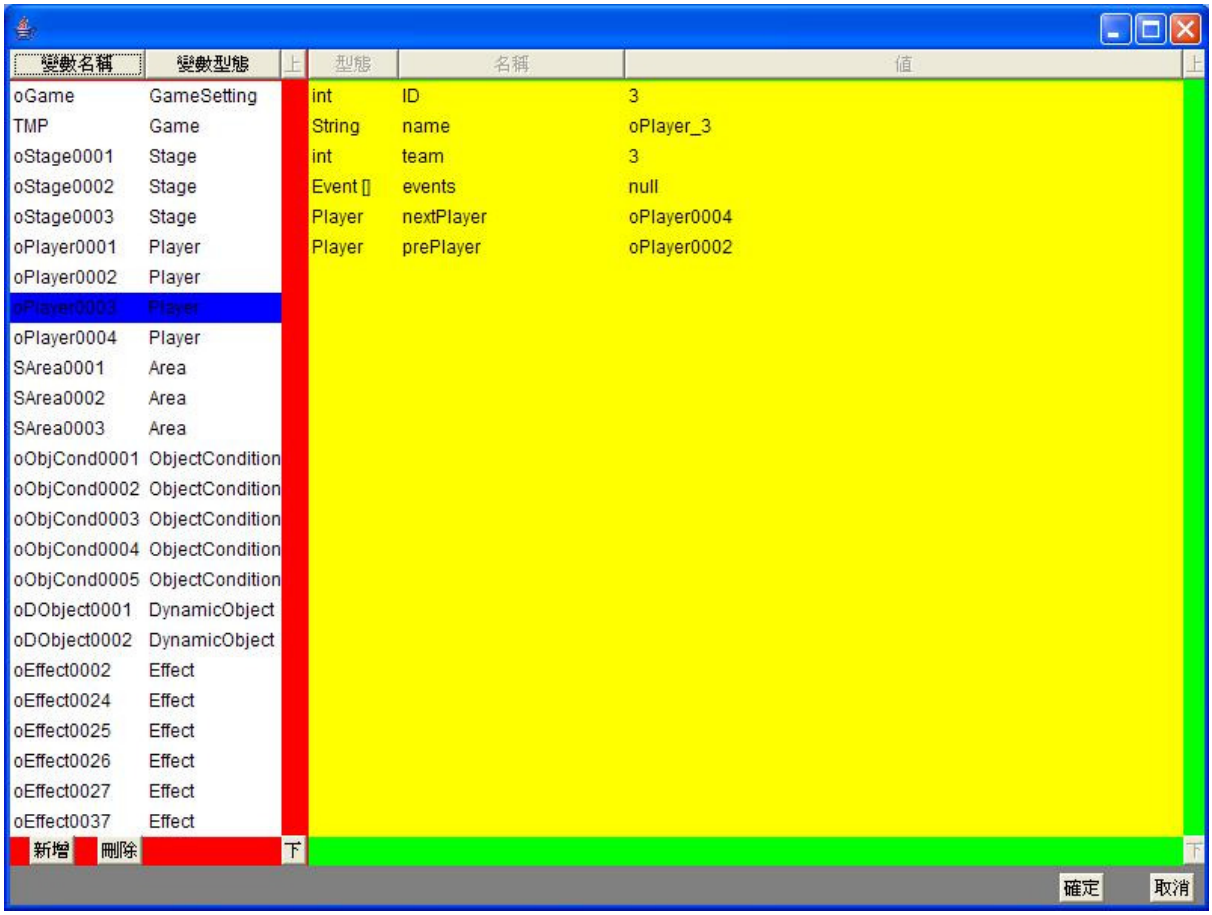


圖 10：物件與動作設定畫面

關於「99 魔法牌」的物件與動作的詳細設定，詳見附錄 B。

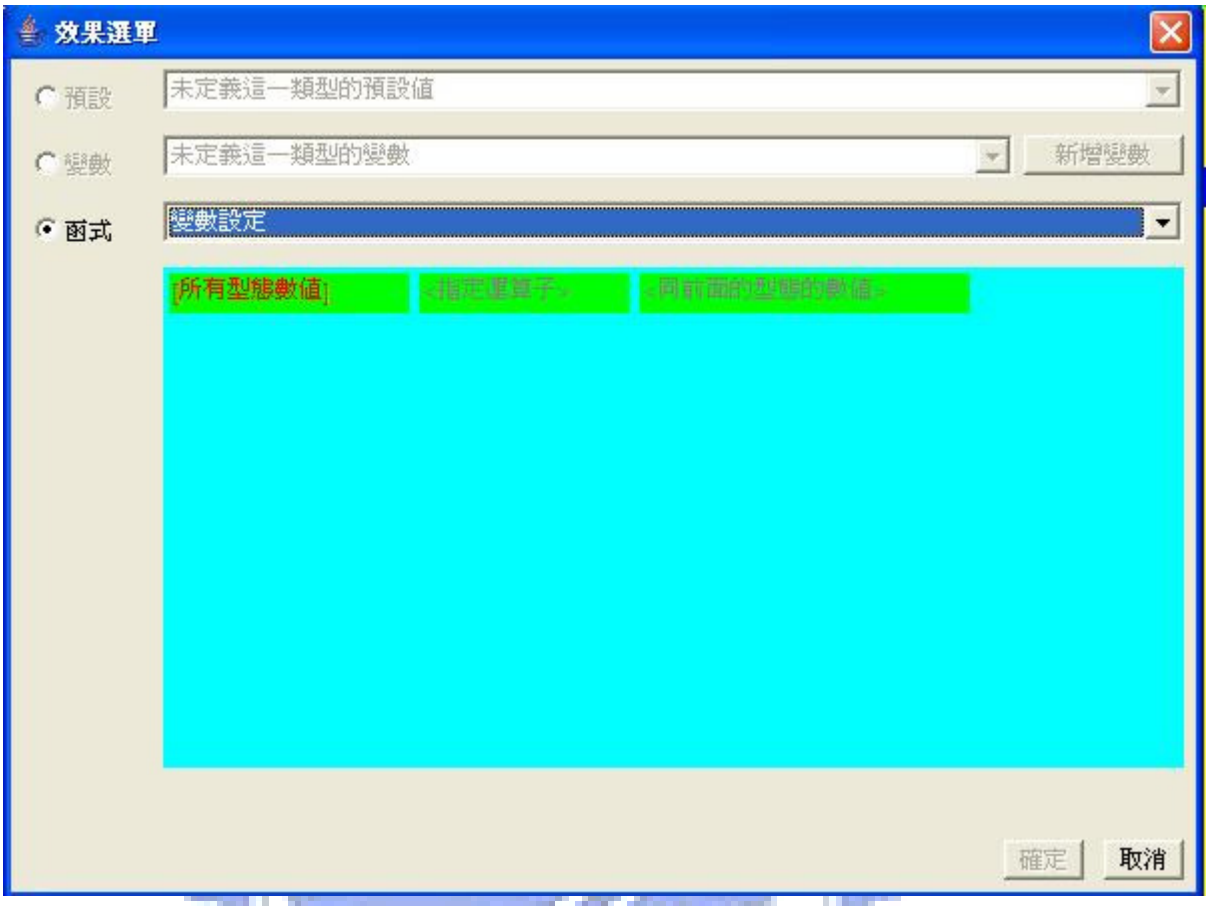


圖 11：使用文法(grammar)設定執行效果畫面

4.7 遊戲擴充

透過設定動作物件與牌卡物件，遊戲設計師可以自行增加遊戲規則，但由於目前系統所支援的「執行效果」的格式種類並不多，尚沒有提供許多較高級的程式設計技巧，所以可能有一些執行效果需要用比較複雜的方式來做設計。所以我們亦在「執行效果」的部份設計了一項「函式呼叫」，然後允許玩家編輯一個擴充用的函式檔，魔法牌遊戲開發系統在執行的時候，將載入玩家所編輯的擴充函式檔，讓玩家可以在執行效果的地方，透過「函式呼叫」這個指令集來存取。關於設定的方式，詳見附錄 C。

第五章、結論與未來展望

本論文提出了魔法牌的明確定義與系統架構，並且實際做出了一個可以開發魔法牌遊戲的魔法牌遊戲開發系統。這個系統以 Java 語言為基礎，透過讀取指令集的方式包裝程式碼，亦允許遊戲設計師自行寫擴充的函式供系統使用。我們所提出的系統，就像是一種包裝在某種程式語言(在此使用 Java)的程式語言一般，故透過該系統，遊戲設計師可以製作各式各樣的魔法牌遊戲。

因為魔法牌遊戲的規則本身就複雜，所以就算是透過該系統來製作魔法牌遊戲，仍然需要花上不少工夫。並且，一款遊戲免不了會需要使用者操作介面(GUI)，但因為各種不同的魔法牌遊戲所需要顯示與操作畫面差別性都非常的大，所以本論文在這方面並沒有做深入的研究。

本系統仍有許多部份是可以繼續加強的：

1. 支援更多更方便使用的程式語言技巧。
2. 支援更多更方便使用的內建函式。
3. 支援圖形化介面的「使用者操作介面編輯器」，讓遊戲設計師可以更方便製作自己所需要的使用者操作介面。
4. 更方便的遊戲規則編輯器，讓遊戲設計師可以更有效率地透過本系統來實作魔法牌遊戲。

另外，因為魔法牌遊戲亦是個規則複雜且平衡性不易設計的遊戲，所以至於魔法牌遊戲的規則平衡性、甚至是 AI 的研究，都會是不錯的相關研究主題。

附錄 A 執行條件與執行效果的文法

因為動作物件的執行條件與執行效果都需要透過文法來產生符合正確格式的設定參數，而這些設定參數又有可能是玩家所設定的類別屬性值，譬如「牌卡.數字」這就是「99 魔法牌」這個遊戲所需要用到的一個自定的屬性值，所以每個遊戲的文法的亦會有所不同，在此我們列出的是動態產生後的 99 魔法牌規則設定文法。

(條件選單)　回傳型態:條件　　變數型態:　　預設值: true | false
函式設定(條件設定: [所有型態數值] <邏輯運算子> <同前面的型態的數值>)
函式設定(複合條件: [複合條件])
函式設定(條件函式: [條件函式])

[複合條件]　回傳型態:條件　　變數型態:條件　　預設值:
函式設定(AND: [複合條件] AND [複合條件])
函式設定(OR:　[複合條件] OR　[複合條件])

[條件]　回傳型態:條件　　變數型態:條件　　預設值: true | false
函式設定(布林值: [布林])

[所有型態數值]　回傳型態:等候上傳　　變數型態:全部　　預設值:
函式設定(字串型態數值: [字串])
函式設定(整數型態數值: [整數])
函式設定(布林型態數值: [布林])
函式設定(遊戲型態數值: [遊戲])
函式設定(階段型態數值: [階段])
函式設定(玩家型態數值: [玩家])
函式設定(牌區型態數值: [牌區])
函式設定(牌卡型態數值: [牌卡])
函式設定(事件型態數值: [事件])

(效果選單)　回傳型態:效果　　變數型態:　　預設值:
函式設定(變數設定:　　[所有型態數值] <指定運算子> <同前面的型態的數值>)
函式設定(if:　　if [條件] then [效果])
函式設定(if-else:　　if [條件] then [效果] else [效果])

函式設定(控制權:　　將控制權交給 [控制權設定])
函式設定(顯示文字:　　顯示文字： [字串])
函式設定(For Loop:　　for [For 迴圈專用變數] = [整數] To [整數] do [效果])
函式設定(While Loop:　while [條件] then [效果])
函式設定(複合效果:　　[複合效果])
函式設定(效果函式:　　[效果型態函式])

[複合效果]　回傳型態:效果　　變數型態:效果　　預設值:
函式設定(AND: [複合效果] AND [複合效果])
函式設定(OR : [複合效果] OR　[複合效果])

// [物件型態.屬性]
==========
[遊戲.屬性]　回傳型態:等候上傳　　變數型態:　　預設值:
函式設定(遊戲.遊戲變數: [遊戲.遊戲])
函式設定(遊戲.階段變數: [遊戲.階段])
函式設定(遊戲.玩家變數: [遊戲.玩家])
函式設定(遊戲.牌區變數: [遊戲.牌區])
函式設定(遊戲.牌卡變數: [遊戲.牌卡])
函式設定(遊戲.事件變數: [遊戲.事件])
函式設定(遊戲.字串變數: [遊戲.字串])
函式設定(遊戲.整數變數: [遊戲.整數])
函式設定(遊戲.布林變數: [遊戲.布林])

[階段.屬性]　回傳型態:等候上傳　　變數型態:　　預設值:
函式設定(階段.遊戲變數: [階段.遊戲])
函式設定(階段.階段變數: [階段.階段])
函式設定(階段.玩家變數: [階段.玩家])
函式設定(階段.牌區變數: [階段.牌區])
函式設定(階段.牌卡變數: [階段.牌卡])
函式設定(階段.事件變數: [階段.事件])
函式設定(階段.字串變數: [階段.字串])
函式設定(階段.整數變數: [階段.整數])
函式設定(階段.布林變數: [階段.布林])

[玩家.屬性]　回傳型態:等候上傳　　變數型態:　　預設值:
函式設定(玩家.遊戲變數: [玩家.遊戲])
函式設定(玩家.階段變數: [玩家.階段])

函式設定(玩家.玩家變數: [玩家.玩家])
函式設定(玩家.牌區變數: [玩家.牌區])
函式設定(玩家.牌卡變數: [玩家.牌卡])
函式設定(玩家.事件變數: [玩家.事件])
函式設定(玩家.字串變數: [玩家.字串])
函式設定(玩家.整數變數: [玩家.整數])
函式設定(玩家.布林變數: [玩家.布林])

[牌區.屬性]　回傳型態:等候上傳　　變數型態:　　預設值:
函式設定(牌區.遊戲變數: [牌區.遊戲])
函式設定(牌區.階段變數: [牌區.階段])
函式設定(牌區.玩家變數: [牌區.玩家])
函式設定(牌區.牌區變數: [牌區.牌區])
函式設定(牌區.牌卡變數: [牌區.牌卡])
函式設定(牌區.事件變數: [牌區.事件])
函式設定(牌區.字串變數: [牌區.字串])
函式設定(牌區.整數變數: [牌區.整數])
函式設定(牌區.布林變數: [牌區.布林])

[牌卡.屬性]　回傳型態:等候上傳　　變數型態:　　預設值:
函式設定(牌卡.遊戲變數: [牌卡.遊戲])
函式設定(牌卡.階段變數: [牌卡.階段])
函式設定(牌卡.玩家變數: [牌卡.玩家])
函式設定(牌卡.牌區變數: [牌卡.牌區])
函式設定(牌卡.牌卡變數: [牌卡.牌卡])
函式設定(牌卡.事件變數: [牌卡.事件])
函式設定(牌卡.字串變數: [牌卡.字串])
函式設定(牌卡.整數變數: [牌卡.整數])
函式設定(牌卡.布林變數: [牌卡.布林])

[事件.屬性]　回傳型態:等候上傳　　變數型態:　　預設值:
函式設定(事件.遊戲變數: [事件.遊戲])
函式設定(事件.階段變數: [事件.階段])
函式設定(事件.玩家變數: [事件.玩家])
函式設定(事件.牌區變數: [事件.牌區])
函式設定(事件.牌卡變數: [事件.牌卡])
函式設定(事件.事件變數: [事件.事件])
函式設定(事件.字串變數: [事件.字串])

函式設定(事件.整數變數: [事件.整數])
函式設定(事件.布林變數: [事件.布林])

// [物件型態.所有型態]
==========
// [遊戲]
[遊戲.遊戲]　回傳型態:遊戲　變數型態:　預設值:
[遊戲.階段]　回傳型態:階段　變數型態:　預設值:
[遊戲.玩家]　回傳型態:玩家　變數型態:　預設值: nowPlayer
[遊戲.牌區]　回傳型態:牌區　變數型態:　預設值:
[遊戲.牌卡]　回傳型態:牌卡　變數型態:　預設值:
[遊戲.事件]　回傳型態:事件　變數型態:　預設值:
[遊戲.字串]　回傳型態:字串　變數型態:　預設值:
[遊戲.整數]　回傳型態:整數　變數型態:　預設值: sum
[遊戲.布林]　回傳型態:布林　變數型態:　預設值:

// [階段]
[階段.遊戲]　回傳型態:遊戲　變數型態:　預設值:
[階段.階段]　回傳型態:階段　變數型態:　預設值: nextStage
[階段.玩家]　回傳型態:玩家　變數型態:　預設值:
[階段.牌區]　回傳型態:牌區　變數型態:　預設值:
[階段.牌卡]　回傳型態:牌卡　變數型態:　預設值:
[階段.事件]　回傳型態:事件　變數型態:　預設值: events
[階段.字串]　回傳型態:字串　變數型態:　預設值: name
[階段.整數]　回傳型態:整數　變數型態:　預設值: ID
[階段.布林]　回傳型態:布林　變數型態:　預設值:

// [玩家]
[玩家.遊戲]　回傳型態:遊戲　變數型態:　預設值:
[玩家.階段]　回傳型態:階段　變數型態:　預設值:
[玩家.玩家]　回傳型態:玩家　變數型態:　預設值: nextPlayer | prePlayer
[玩家.牌區]　回傳型態:牌區　變數型態:　預設值: areas
[玩家.牌卡]　回傳型態:牌卡　變數型態:　預設值:
[玩家.事件]　回傳型態:事件　變數型態:　預設值: events
[玩家.字串]　回傳型態:字串　變數型態:　預設值: name
[玩家.整數]　回傳型態:整數　變數型態:　預設值: ID | team
[玩家.布林]　回傳型態:布林　變數型態:　預設值:

// [牌區]
[牌區.遊戲]　回傳型態:遊戲　變數型態:　預設值:
[牌區.階段]　回傳型態:階段　變數型態:　預設值:
[牌區.玩家]　回傳型態:玩家　變數型態:　預設值:
[牌區.牌區]　回傳型態:牌區　變數型態:　預設值:
[牌區.牌卡]　回傳型態:牌卡　變數型態:　預設值: cards
[牌區.事件]　回傳型態:事件　變數型態:　預設值: events
[牌區.字串]　回傳型態:字串　變數型態:　預設值: name
[牌區.整數]　回傳型態:整數　變數型態:　預設值: ID | viewPlayer
[牌區.布林]　回傳型態:布林　變數型態:　預設值:

// [牌卡]
[牌卡.遊戲]　回傳型態:遊戲　變數型態:　預設值:
[牌卡.階段]　回傳型態:階段　變數型態:　預設值:
[牌卡.玩家]　回傳型態:玩家　變數型態:　預設值:
[牌卡.牌區]　回傳型態:牌區　變數型態:　預設值:
[牌卡.牌卡]　回傳型態:牌卡　變數型態:　預設值:
[牌卡.事件]　回傳型態:事件　變數型態:　預設值: events
[牌卡.字串]　回傳型態:字串　變數型態:　預設值: name
[牌卡.整數]　回傳型態:整數　變數型態:　預設值: ID | number
[牌卡.布林]　回傳型態:布林　變數型態:　預設值:

// [事件]
[事件.遊戲]　回傳型態:遊戲　變數型態:　預設值:
[事件.階段]　回傳型態:階段　變數型態:　預設值:
[事件.玩家]　回傳型態:玩家　變數型態:　預設值:
[事件.牌區]　回傳型態:牌區　變數型態:　預設值:
[事件.牌卡]　回傳型態:牌卡　變數型態:　預設值:
[事件.事件]　回傳型態:事件　變數型態:　預設值:
[事件.字串]　回傳型態:字串　變數型態:　預設值: name | note | type | execType | actionTriggerType
[事件.整數]　回傳型態:整數　變數型態:　預設值: ID
[事件.布林]　回傳型態:布林　變數型態:　預設值:

// [型態]
==========
[遊戲]　回傳型態:遊戲　變數型態:遊戲　預設值:
函式設定(遊戲.遊戲: [遊戲] . [遊戲.遊戲])

函式設定(階段.遊戲: [階段] . [階段.遊戲])
函式設定(玩家.遊戲: [玩家] . [玩家.遊戲])
函式設定(牌區.遊戲: [牌區] . [牌區.遊戲])
函式設定(牌卡.遊戲: [牌卡] . [牌卡.遊戲])
函式設定(事件.遊戲: [事件] . [事件.遊戲])
函式設定(遊戲型態函式: [遊戲型態函式])

[階段]　回傳型態:階段　變數型態:階段　預設值:
函式設定(遊戲.階段: [遊戲] . [遊戲.階段])
函式設定(階段.階段: [階段] . [階段.階段])
函式設定(玩家.階段: [玩家] . [玩家.階段])
函式設定(牌區.階段: [牌區] . [牌區.階段])
函式設定(牌卡.階段: [牌卡] . [牌卡.階段])
函式設定(事件.階段: [事件] . [事件.階段])
函式設定(階段型態函式: [階段型態函式])

[玩家]　回傳型態:玩家　變數型態:玩家　預設值:
函式設定(遊戲.玩家: [遊戲] . [遊戲.玩家])
函式設定(階段.玩家: [階段] . [階段.玩家])
函式設定(玩家.玩家: [玩家] . [玩家.玩家])
函式設定(牌區.玩家: [牌區] . [牌區.玩家])
函式設定(牌卡.玩家: [牌卡] . [牌卡.玩家])
函式設定(事件.玩家: [事件] . [事件.玩家])
函式設定(玩家型態函式: [玩家型態函式])

[牌區]　回傳型態:牌區　變數型態:牌區　預設值:
函式設定(遊戲.牌區: [遊戲] . [遊戲.牌區])
函式設定(階段.牌區: [階段] . [階段.牌區])
函式設定(玩家.牌區: [玩家] . [玩家.牌區])
函式設定(牌區.牌區: [牌區] . [牌區.牌區])
函式設定(牌卡.牌區: [牌卡] . [牌卡.牌區])
函式設定(事件.牌區: [事件] . [事件.牌區])
函式設定(牌區型態函式: [牌區型態函式])

[牌卡]　回傳型態:牌卡　變數型態:牌卡　預設值:
函式設定(遊戲.牌卡: [遊戲] . [遊戲.牌卡])
函式設定(階段.牌卡: [階段] . [階段.牌卡])
函式設定(玩家.牌卡: [玩家] . [玩家.牌卡])

函式設定(牌區.牌卡: [牌區] . [牌區.牌卡])
函式設定(牌卡.牌卡: [牌卡] . [牌卡.牌卡])
函式設定(事件.牌卡: [事件] . [事件.牌卡])
函式設定(牌卡型態函式: [牌卡型態函式])

[事件]　回傳型態:事件　變數型態:事件　預設值:
函式設定(遊戲.事件: [遊戲] . [遊戲.事件])
函式設定(階段.事件: [階段] . [階段.事件])
函式設定(玩家.事件: [玩家] . [玩家.事件])
函式設定(牌區.事件: [牌區] . [牌區.事件])
函式設定(牌卡.事件: [牌卡] . [牌卡.事件])
函式設定(事件.事件: [事件] . [事件.事件])
函式設定(事件型態函式: [事件型態函式])

[字串]　回傳型態:字串　變數型態:字串　預設值:
函式設定(遊戲.字串: [遊戲] . [遊戲.字串])
函式設定(階段.字串: [階段] . [階段.字串])
函式設定(玩家.字串: [玩家] . [玩家.字串])
函式設定(牌區.字串: [牌區] . [牌區.字串])
函式設定(牌卡.字串: [牌卡] . [牌卡.字串])
函式設定(事件.字串: [事件] . [事件.字串])
函式設定(字串型態函式: [字串型態函式])

[整數]　回傳型態:整數　變數型態:整數　預設值:
函式設定(遊戲.整數: [遊戲] . [遊戲.整數])
函式設定(階段.整數: [階段] . [階段.整數])
函式設定(玩家.整數: [玩家] . [玩家.整數])
函式設定(牌區.整數: [牌區] . [牌區.整數])
函式設定(牌卡.整數: [牌卡] . [牌卡.整數])
函式設定(事件.整數: [事件] . [事件.整數])
函式設定(整數型態函式: [整數型態函式])

[布林]　回傳型態:布林　變數型態:布林　預設值:
函式設定(遊戲.布林: [遊戲] . [遊戲.布林])
函式設定(階段.布林: [階段] . [階段.布林])
函式設定(玩家.布林: [玩家] . [玩家.布林])
函式設定(牌區.布林: [牌區] . [牌區.布林])
函式設定(牌卡.布林: [牌卡] . [牌卡.布林])

函式設定(事件.布林: [事件] . [事件.布林])
函式設定(布林型態函式: [布林型態函式])

[字串型態函式]　回傳型態:字串　變數型態:　預設值:
函式設定(字串+字串: [字串] # [字串])

[整數型態函式]　回傳型態:整數　變數型態:　預設值:
函式設定(整數計算: [[整數] [整數算術運算子] [整數]])

[效果]　回傳型態:效果　變數型態:效果　預設值: 無效果
[控制權設定] 回傳型態:　變數型態:　預設值: SYSTEM | PLAYER
[For 迴圈專用變數]　回傳型態:整數　變數型態:　預設值: FOR_1 | FOR_2 | FOR_3 | FOR_4 | FOR_5 | FOR_6 | FOR_7 | FOR_7 | FOR_8 | FOR_9
[物件條件]　回傳型態:等候上傳　變數型態:物件型態:全部　預設值:
[整數算術運算子] 回傳型態:　變數型態:　預設值: + | - | * | / | %
[遊戲型態函式]　回傳型態:遊戲　變數型態:　預設值: 這個遊戲
[階段型態函式]　回傳型態:階段　變數型態:　預設值:
[玩家型態函式]　回傳型態:玩家　變數型態:　預設值: 自己 | 對手(限兩人時) | 所有玩家 | 所有對手 | 所有同盟
[牌區型態函式]　回傳型態:牌區　變數型態:　預設值:
[牌卡型態函式]　回傳型態:牌卡　變數型態:　預設值: 這張牌卡
[效果型態函式]　回傳型態:效果　變數型態:　預設值: 暫停
[數字邏輯運算子] 回傳型態:　變數型態:　預設值: > | >= | == | <= | < | !=
[非數字邏輯運算子]　回傳型態:　變數型態:　預設值: == | !=
[數字指定運算子] 回傳型態:　變數型態:　預設值: = | += | -= | *= | /= | %=
[非數字指定運算子]　回傳型態:　變數型態:　預設值: =

<同前面的型態的數值>　回傳型態:　變數型態:同前　預設值:
if (preType==字串) [字串]
if (preType==整數) [整數]
if (preType==布林) [布林]
if (preType==遊戲) [遊戲]
if (preType==階段) [階段]
if (preType==玩家) [玩家]
if (preType==牌區) [牌區]
if (preType==牌卡) [牌卡]
if (preType==事件) [事件]
END

<邏輯運算子>　回傳型態:　變數型態:　預設值:
if (preType==整數)　[數字邏輯運算子]
[非數字邏輯運算子]
END

<指定運算子>　回傳型態:　變數型態:　預設值:
if (preType==整數)　[數字指定運算子]
[非數字指定運算子]
END

附錄 B 「99 魔法牌」物件設定

- 遊戲設定

String	*boolean*	*Stage*	*Player[]*	*Action[]*	*int*	*Player*
Name	**isGameover**	**Number**	**Players**	**actions**	**number**	**nowPlayer**
99 魔法牌	false	換誰階段	玩家 1 玩家 2	無	0	玩家 2

- 階段設定

Int	*String*	*Stage*	*Action[]*
ID	**name**	**nextStage**	**actions**
1	換誰階段	出牌階段	『顯示現在階段名稱』 『決定換誰』 『顯示現在輪到誰』
2	出牌階段	抽牌階段	『顯示現在階段名稱』 『輪到的玩家敗北判斷』 『顯示目前遊戲積數』 『讓現在輪到的玩家可以行動』 『交出控制權給玩家』
3	抽牌階段	換誰階段	『顯示現在階段名稱』 『輪到的玩家抽一張牌』

- 玩家設定

int	*String*	*Player[]*	*Player[]*	*Areas[]*	*Action[]*	*Player*	*Player*
ID	**name**	**allys**	**opponents**	**areas**	**actions**	**上家**	**下家**
1	玩家 1	無	玩家 2	牌庫 1 手上 1 廢牌 1	無	玩家 2	玩家 2
2	玩家 2	無	玩家 1	牌庫 2 手上 2 廢牌 2	無	玩家 1	玩家 1

註：canDoAction 屬性值，全部都爲 false

- 牌區設定

int	*String*	*Int*	*Card[]*	*Action[]*
ID	**name**	**viewType**	**cards**	**actions**
1	牌庫 1	0(無)	LOAD	無
2	手上 1	1(自己)	LOAD	無
3	廢牌 1	7(全)	LOAD	無
4	牌庫 2	0(無)	LOAD	無
5	手上 2	1(自己)	LOAD	無
6	廢牌 2	7(全)	LOAD	無

- 牌卡設定

int	*String*	*Action[]*	*Int*	*Boolean*
ID	**name**	**actions**	**Number**	**special**
1	A	『加』	1	false
2	2	『加』	2	false
3	3	『加』	3	false
4	4	『加』	4	false
5	5	『加』	5	false
6	6	『加』	6	false
7	7	『加』	7	false
8	8	『加』	8	false
9	9	『加』	9	false
10	10	『加』、『減』	10	true
11	J	『pass』	0	true
12	Q	『加』、『減』	20	true
13	K	『99』	0	true
14	Joker1	『丟牌』	0	true
15	Joker2	『再出』	0	true

- 動作設定

因為動作物件的設定全部都是動態設定的，所以在 timings、conditions、effects 的部份，格式一律為<type>parameter，譬如『顯示現在階段名稱』這個動作，他的 conditions 是<布林值>true，意思就是他的 conditions 物件的 type=”布林值”，parameter=”true”

※階段動作

	ID	Name	triggerType
	1	『顯示現在階段名稱』	無
Timings	<系統呼叫>無		
Conditions	<布林值>true		
Effects	<顯示文字>現在階段：[遊戲.現在的階段.名稱]		

	ID	Name	triggerType
	2	『決定換誰』	無
Timings	<系統呼叫>無		
Conditions	<布林值>true		
Effects	<變數指定>[遊戲.輪到的玩家] = [遊戲.輪到的玩家.下家]		

	ID	name	triggerType
	3	『顯示現在輪到誰』	無
Timings	<系統呼叫>無		
Conditions	<布林值>true		
Effects	<顯示文字>現在輪到[遊戲.輪到的玩家.名稱]		

	ID	name	triggerType
	4	『輪到的玩家敗北判斷』	無
Timings	<系統呼叫>無		
Conditions	<布林值>true		
Effects	<函式呼叫>玩家敗北判斷([遊戲.輪到的玩家])		

	ID	name	triggerType
	5	『顯示目前遊戲積數』	無
Timings	<系統呼叫>無		
Conditions	<布林值>true		
Effects	<顯示文字>遊戲積數：[遊戲.遊戲積數]		

	ID	name	triggerType
	6	『讓現在輪到的玩家可以行動』	無
Timings	<系統呼叫>無		
Conditions	<布林值>true		
Effects	<變數指定>[遊戲.輪到的玩家.可以行動] = true		

	ID	name	triggerType
	7	『交出控制權給玩家』	無
Timings	<系統呼叫>無		
Conditions	<布林值>true		
Effects	<控制權>玩家		

	ID	name	triggerType
	8	『輪到的玩家抽一張牌』	無
Timings	<系統呼叫>無		
Conditions	<布林值>true		
Effects	<函式呼叫>[遊戲.輪到的玩家].抽牌(1)		

※牌卡動作

	ID	name	triggerType
	9	『加』	加
Timings	<玩家使用>出牌階段		
Conditions	<運算式>[這張牌.所在位置] == [這張牌.擁有玩家.手上] & <運算式> [這張牌. 擁有玩家.可以行動] == true & <運算式> [遊戲.遊戲積數] + [這張牌.數字] <= 99		
Effects	<函式呼叫>[這張牌].移動到([這張牌.擁有玩家.廢牌] & <變數指定>[遊戲.遊戲積數] += [這張牌.數字] & <變數指定>[這張牌. 擁有玩家.可以行動] = false		

	ID	name	triggerType
	10	『減』	減
Timings	<玩家使用>出牌階段		
Conditions	<運算式>[這張牌.所在位置] == [這張牌.擁有玩家.手上] & <運算式> [這張牌. 擁有玩家.可以行動] == true & <運算式> [遊戲.遊戲積數] - [這張牌.數字] >= 0		
Effects	<函式呼叫>[這張牌].移動到([這張牌.擁有玩家.廢牌] & <變數指定>[遊戲.遊戲積數] -= [這張牌.數字] & <變數指定>[這張牌. 擁有玩家.可以行動] = false		

	ID	name	triggerType
	11	『99』	等於
Timings	<玩家使用>出牌階段		
Conditions	<運算式>[這張牌.所在位置] == [這張牌.擁有玩家.手上] & <運算式> [這張牌. 擁有玩家.可以行動] == true		
Effects	<函式呼叫>[這張牌].移動到([這張牌.擁有玩家.廢牌] & <變數指定>[遊戲.遊戲積數] = 99 & <變數指定>[這張牌. 擁有玩家.可以行動] = false		

	ID	name	triggerType
	12	『pass』	無
Timings	<玩家使用>出牌階段		
Conditions	<運算式>[這張牌.所在位置] == [這張牌.擁有玩家.手上] & <運算式> [這張牌. 擁有玩家.可以行動] == true		
Effects	<函式呼叫>[這張牌].移動到([這張牌.擁有玩家.廢牌] & <變數指定>[這張牌. 擁有玩家.可以行動] = false		

	ID	name	triggerType
	13	『丟牌』	無
Timings	<動作結束觸發>減		
Conditions	<運算式>[這張牌.所在位置] == [這張牌.擁有玩家.手上] & <運算式>[這張牌.擁有玩家] != [這張牌.觸發牌.擁有玩家]		
Effects	<函式呼叫>[這張牌].移動到([這張牌.擁有玩家.廢牌]) & <函式呼叫>[這張牌.擁有玩家].抽牌(1) & <函式呼叫>[這張牌.觸發牌.擁有玩家].棄牌(1)		

	ID	Name	triggerType
	14	『再抽』	無
Timings	<動作開始觸發>減		
Conditions	<運算式>[這張牌.所在位置] == [這張牌.擁有玩家.手上] & <運算式>[這張牌.擁有玩家] != [這張牌.觸發牌.擁有玩家]		
Effects	<函式呼叫>[這張牌].移動到([這張牌.擁有玩家.廢牌]) & <函式呼叫>[這張牌.擁有玩家].抽牌(1) & <變數指定>[遊戲.遊戲積數] -= 5 & <函式呼叫>[這張牌.觸發牌.擁有玩家].出牌(1) & <函式呼叫>[這張牌.觸發牌.擁有玩家].抽牌(1)		

附錄C 其他程式相關說明文件

一、 執行「魔法牌遊戲開發系統」的方式

1. 執行 menuFrame.java

讀入檔案：

GameSetting.txt
varList
otherFunction.java

產生檔案：

GameSetting.txt
varList
objList.txt
server.config

2. 執行 gameGen.java

讀入檔案：

objList.txt

產生檔案：

TMP.java
ObjectCondition.java
DynamicObject.java
Effect.java
Condition.java
Timing.java
Action.java
Card.java
Area.java

Player.java
Stage.java
Game.java

3. 編譯遊戲檔（compile 以下 Java 檔）

TMP.java
ObjectCondition.java
DynamicObject.java
Effect.java
Condition.java
Timing.java
Action.java
Card.java
Area.java
Player.java
Stage.java
Game.java

二、 執行遊戲的方式

1. 執行 server

執行 server.java

讀入檔案：

server.config

2. 執行 client

執行 client.java

讀入檔案：

playerData.txt（玩家的牌組資訊，為方便測試，暫由 client 端提供）

三、 遊戲執行流程

由於魔法牌遊戲的玩家資訊與玩家所使用的牌組資訊會依玩家所使用的牌組有關，所以遊戲會需要於所有的玩家都建立好連線之後，先讀入玩家資訊與牌組資訊，然後再動態產生遊戲物件。所以遊戲會透過 Client 來呼叫執行，遊戲程式 (Game)的執行程序如下：

```
Client.startGame()        // 由 client 呼叫 startGame()
    game.loadGameSetting() ;          // 讀入遊戲的組成設定
    game.setPlayerData(PlayerData) ; // 讀入玩家資料與牌組資料
    game.setup() ;                    // 遊戲建置
        game.setObj() :               // 動態產生所有遊戲物件
        game.setAllAction() :         // 設定所有的 action 做好 link
        game.setAllObject() :         // 設定所有的 object 做好 link
        game.setOwner() :             // 設定所有物件的 owner link
        game.setTiming() :            // 設定所有動作的 Timing
        game.setInit() ;              // 初始遊戲的一些內部設定值
    game.run() ;                      // 開始遊戲
        while(isGameover) {
            game.startThisStage() ;   // 進入目前所在的階段
            game.doThisStage() ;      // 執行目前所在的階段
            game.endThisStage() ;     // 結束目前所在的階段
            game.goNextStage() ;      // 進入下一個階段
        }
        game.setGameover() ;          // 遊戲結束處理
```

四、 執行動作的方式

遊戲裡的動作有以下兩種執行的方式：

1. 由階段執行某個動作
2. 由使用者選擇動作執行

其他相關的要點如下：

- 在執行動作時，為了顧慮到有時會有很多動作一起被執行，所以我們使用一個佇列(Queue):vActionQueue 來做控制，將所有要被執行的動作先依序加入這個佇列中，然後再在要「結算」的時候，依序執行佇列裡的每個動作。
- 執行動作時，會呼叫 execAction(Action action)將動作放入 vActionQueue 裡，然後，再呼叫 execActionQueue()來依序執行 vActionQueue 裡的每個動作。
- 呼叫 execEventQueue()的時機有兩者：
  1. 在呼叫 execAction(Action action)時，傳入的動作的 execType 屬性值為"exec"。
  2. 階段在將所有的階段動作放入 vActionQueue 後呼叫。
- 執行動作的流程如下：

```
if (canUse()) {
        doPreAction();  // 執行這個動作的動作前觸發動作
        doThisAction(); // 執行這個動作本身的動作
        doPostAction(); // 執行這個動作的動作後觸發動作
}
```

五、 魔法牌遊戲開發系統的執行流程

我們所設計的魔法牌遊戲開發系統的執行流程如下：

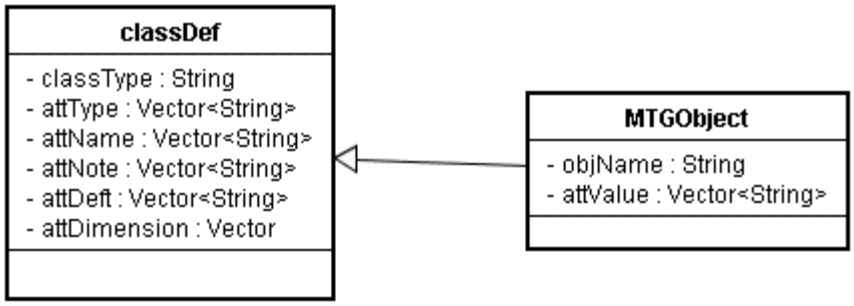
```
menuFrame.dataInit()            // 設定資料
    readGameSetting() ;
    genClassDef() ;
    genObject() ;
    readVarList() ;
    resetVarList() ;
    genGrammar() ;
    readGrammar() ;
    setFunction() ;
    resetGrammar() ;
gameFrame.GUIInit()             // 設定操作介面資料
```

- readGameSetting()
  讀入遊戲設定檔，包含遊戲架構、牌組設定、玩家自定的類別定義。
- genClassDef()
  產生遊戲會用到的「類別」資料。
- genObject()
  產生遊戲所需要的「物件」資料。
- readVarList()
  讀入遊戲的物件清單，存入「物件列表」。
- resetVarList()
  更新遊戲的「物件列表」(若讀入的物件清單裡有不正確的物件則刪除，如玩家人數應該只有兩人，但讀入的物件列表有三個玩家物件，便要將多餘的玩家物件刪除)
- genGrammar()
  依據使用者對整個遊戲的設定，產生「條件」與「效果」設

定所需要用的「文法」資訊。該資訊會存成檔案，方便遊戲設計者閱讀。

- readGrammar()
  讀入剛才所產生的「文法」檔案並建立資料結構。
- setFunction()
  讀入「函式擴充」檔，使用者定義的函式會於此讀入。
- resetGrammar()
  清除「文法」檔案中不會被存取到的文法規則。

另外，在該程式裡，我們主要使用 classDef 來儲存「類別」資料，使用 MTGObject 來儲存「物件」資料。這兩個類別的資料結構如下圖所示：



各個參數的詳細說明如下：

名稱	型態	說明
classType	String	類別名稱
objName	String	物件名再
attType	Vector<String>	屬性值型態
attName	Vector<String>	屬性值名稱
attNote	Vector<String>	屬性值註解
attValue	Vector<String>	屬性值實際的值
attDeft	Vector<String>	屬性值預設值
attDimension	Vector	屬性值的維度

六、 函式擴充

遊戲設計師可視需要自行寫函式(程式碼),然後擴充到系統裡。系統會讀入 otherFunction.java 這個程式，並依據正確的格式將函式資料匯入系統裡。otherFunction.java 裡的格式如下：

```
class Game {
      /**
       *
       */
      void getPlayer() {
            ....
      } // end function
}

class Stage {
}

class Player {
}
....
```

otherFunction.java 由數個 class 所組成，這些 class 即遊戲裡會用到的所有類別。設計者可視需要將自己的 function 寫在合適的 class 下面。自定 function 請依下列格式設定：

1. 註解區：

```
/**
 * 註解
 */
```

由 ”/** ” 開頭，由” */ “結尾。就算該函式沒有註解，仍然需要有註解區。

2. 函式區：

```
retType funcName(parameterList)
{
    // code
} // end function
```

函式一定需要回傳型態(retType)、函式名稱(funcName)，至於函式的參數列表(parameterList)可有可沒有。函式的左大括號一定要在下一行，右大括號後面一定要接上一個空格，然後再來是單行註解的"end function"。

當 function 符合以下的格式，就可以在 tool 裡的效果設定裡看到自己寫的 function 了。

參考文獻

[1] Wikipedia , “交換卡片遊戲” , available from http://www.wikipedia.org/

[2] Wizard of the Coast , “魔法風雲會” , available from http://www.wizards.com/magic/ , 1995

[3] KONAMI , “遊戲王” , available from http://www.yugioh-card.com/ , 1996

[4] Wizard of the Coast , “魔法風雲會 online” , available from http://www.wizards.com/default.asp?x=magic/magiconline , 2003

[5] KONAMI , “遊戲王 online” , available from http://www.yugioh-online.net/ , 1996

[6] GRAVITY , “仙境傳說 RO” , available from http://ro.gameflier.com/ , 2004

[7] Blizzard , “worldofwarcraft(WOW)” , available from http://www.worldofwarcraft.com/ , 2005

[8] netmarble , “RO-TCG” , available from http://game.netmarble.jp/cpsite/ragnaroktcg/ , 2004~2007

[9] Blizzard , “worldofwarcraft-TCG” , available from http://entertainment.upperdeck.com/wow/en/ , 2006

[10]KOEI , “真・三國無雙” , available from http://www.wikipedia.org/ 查 “真・三國無雙” , 2001

[11]KOEI , “真・三國無雙 4TCG” , available from http://www.gamecity.ne.jp/smusou4/tcard/ , 2005

[12]actreca.com , “鋼彈大戰(GundamWar)” , available from http://www.diana.dti.ne.jp/~hos/ , 1999 (日本) http://www.spp.com.tw/asp/gundamwar/index.asp , 1999(台灣)

[13] 麗嬰國際 , “神奇寶貝 TCG” , available from http://www.funbox.com.tw/pokemontcg/ , 2006

[14]智冠科技 , “中國魔法牌” , available from http://sg.soft-world.com/ , 1998 ~ 2004

[15]Nival Interactive , “蒼穹霸主 2” , available from http://www.etherlords.com/ , 2002

[16]Qmud.com , “永恆之夢” , available from http://www.qmud.com/v3.5/ , 2003 ~ 2005

[17]Dex Entertainment , “神樣的年代記” , available from http://alteil.jp/index.php , 2004

[18]Apus Software , “Astral Master” , available from http://www.astralmasters.com/ , 2004

[19]I-Chen Wu and J. J. Hsu, "The Model and Systems for Play-on-table Games", IEICE Trans. INF. & SYST. (SCI), VOL. E87-D, No. 11, November 2004.

[20]Sun , “Java”, available from http://java.sun.com/ , 1994