遊戲引擎狀態導向暨抽象事件簡易開發架構設計與實作

Design and Implementation of Simple Development Framework Game Engine with State-Oriented and Abstract Game Event

高嘉廷
shih hsin university
mhp055360@hotmail.com

鄭武堯
shih hsin university
wycheng@cc.shu.edu.tw

摘要

本研究提出並實現一種基於遊戲高階狀態導向的開發架構，此架構提供使用者以遊戲系統狀態的流程概念進行遊戲開發，並且幫助實現遊戲事件的群組管理。減少開發者對事件管理的工作量。

本研究架構實現於 Gameplay3D 這款開源遊戲引擎之上，以該引擎架構為基底向上架構一層更高階的使用者開發介面，增加新的引擎功能，並提出一種基於抽象概念的事件宣告，避免跨平台遊戲開發時需要撰寫的多種輸入裝置事件。使遊戲事件的定義與開發方式更為一致。提出一種基於非程式邏輯導向的遊戲物件群組操作，使物件操作更為直覺且更加彈性。針對遊戲外部裝置的連接與開發，提出人機操作辨識與遊戲邏輯各自獨立的開發架構，增加程式碼的可重用性並且利於遊戲的多人協同開發。

關鍵詞：狀態導向遊戲開發、跨平台抽象事件、群組物件管理、人機操作辨識抽離。

Abstract

This study proposes and implements a state-based high-end game development oriented architecture that provides the user with the state of the process concept game system for game development, and help achieve the group management game events. Developers managed to reduce the workload of the event.

This study architecture implemented on top of the open-source game engine Gameplay3D to the engine architecture as the base layer of a higher order structure up to develop a user inter-face, adding new engine features and made the event a declaration based on abstraction, avoid cross-platform when game developers need to write a variety of input devices events. Make more consistent way to define and develop the game of the event. Program logic is proposed based on non-game-oriented group of objects operation, the object operation more intuitive and more flexible. For connection to an external device game development proposed operation identification and game logic separate development framework, increasing code reusability and is conducive to the game's multiplayer collaborative development.

Keywords: *State-oriented game development, Cross-platform abstract event, Object Management Group, Man-machine identification detached.*

一、研究背景與目的

近年來智慧型行動裝置的普及，開拓了行動平台的新興遊戲市場，遊戲引擎的發展也朝著市場潮流在變動，商用引擎開始注重行動平台的效能優化，同時能夠支援更多的硬體外接裝置。在此環境下，遊戲推陳出新的速度急遽增加，如何簡易且快速的開發一款遊戲成為在遊戲市場搶占先機的關鍵。本研究秉持世新大學雲端互動科技實驗室一貫的精神，期望將遊戲開發的複雜度進一步簡化，帶給使用者更容易學習、更加便利的遊戲開發。

本研究透過狀態導向的使用者開發架構，讓遊戲開發初學者能在沒有專案分工經驗的情況下，開發各自獨立的遊戲狀態而不會產生程式碼衝突。改善跨平台遊戲開發時，因應不同平台上需要有不同事件定義的冗長架構，讓初學者可以在不必套用程式邏輯的概念下，對群組物件進行一致性的操作。對於外部硬體裝置的連接，改善其運行架構，達到裝置辨識與遊戲邏輯各自獨立的目的。整合新舊引擎架構的優點，增加新功能於新的引擎核心當中，改善舊有架構疏失，達到易用易學之目的。

二、文獻探討

2.1. Gameplay3D

Gameplay3D[19]是一款開放原始碼的跨平台遊戲引擎，支援 Microsoft Windows、Apple MacOS X、Linux、Apple iOS、Google Android、BlackBerry 六種平台，橫跨桌上型電腦與行動裝置的範疇。具有一般遊戲引擎的基本功能如：2D 繪圖、3D 繪圖、物理效果、動畫播放、粒子特效等，同時支持多種輸入事件，包含：鍵盤事件、滑鼠事件、觸碰事件以及陀螺儀與手勢辨識。提供基本的 Shader 效果供開發者使用，美術方面由 3Ds Max 以及 Maya 進行編輯後透過轉換工具匯入，具有 Scene Graph 樹狀結構來管理各項 Node。遊戲開發語言為 C++，支援 Lua script 的撰寫。除了欠缺高階使用者開發介面以及事件編輯器等引擎功能外，基礎功能已經相當完善，在各平台上的執行效能也可以有不錯的表現，此引擎同時作為本研究之基礎架構平台。

2.2. IS.GD 2.8

IS.GD Game Engine[1]為世新大學雲端互動科技實驗室所開發維護之遊戲引擎，其中 IS.GD 2.8 為用於 PC 平台遊戲開發的遊戲引擎。此版本引擎使用第三方繪圖引擎 Ogre 為基礎架構，以 Nvdia PhysX 作為物理引擎。引擎上層以 XML 階層式架構配合引擎資料驅動，由 Framework XML 統一管理所有場景。使用者可以在 Scene XML 定義場景物件、設定遊戲參數。在 Trigger XML 定義遊戲事件、使用者輸入事件等。Proxy XML 則用來連結管理 User Rule 及 Trigger XML，使用者可以將遊戲事件對應到自行撰寫的 Rule Function。其他還有 MultiInput XML 用來對應各種外部輸入裝置的輸入指令，方便使用者連接外部裝置。

2.3. IS.GD 4.0

IS.GD 4.0[3]是世新大學雲端互動科技實驗室近年來開發的較新版本引擎，用於行動平台的遊戲開發，引擎底層為 GFX Game Engine，開發上支援 Apple iOS 以及 Google Android 兩大平台。由於 GFX Game Engine 本身所提供的功能以及函式庫過於低階，使用者需直接撰寫 OpenGL 函式，且存在執行時期效能低落的問題，不適合應用於遊戲開發。為解決 GFX Game Engine 的各項問題，IS.GD 4.0 版本改善了原先 GFX 的引擎架構，將引擎載入時期的資料運算抽離至設計階段，事先儲存成 Binary 格式，藉此提升執行時期效能，同時補足原先 GFX Game Engine 所欠缺的基礎遊戲函式，包含動畫、物理、網路、物件存取以及場景管理等，重新包裝為高階函式功能。同時建立了與 IS.GD 2.8 相同的 XML 資料驅動層，將遊戲資料的定義自程式碼中抽離，增加開發上設計與修改的彈性。

2.4. 有限狀態機

有限狀態機[17]是一個包含有限集合的狀態，並表示狀態與狀態之間轉移和執行行為的數學模型。其運作概念常用於數位電路的硬體設計與軟體工程的系統設計。對於狀態集合中的各別狀態而言，可能出現的行為包括：進入動作、退出動作、輸入動作以及轉移動作。有限狀態機的運作會先定義出一個起始狀態，當滿足該狀態下的某項條件後將觸發狀態轉移的動作，該觸發條件可能是人為輸入或是程式運算的結果。

2.5. 遊戲狀態導向

一個遊戲的運行從開始到結束，其實經過了多次的狀態轉換，這些狀態是在遊戲開發階段就定義完成，因此這些狀態將會成為一個有限的集合，是一個有限狀態機。對於遊戲狀態的管理，已經有學者提出相關研究。文獻[5]提出了一種基於點對點網路遊戲的狀態與事件管理機制，有助於提升大型分散式網路遊戲的運行效能。文獻[2]提出一種以 XML 格式撰寫的遊戲狀態模板，透過宣告狀態的結構與屬性，轉換成遊戲內的事件，可以做到重覆宣告狀態事件於不同遊戲物件與對象，減少撰寫類似事件的程式碼長度。

2.6. 遊戲簡易開發機制

遊戲開發一直以來都不是一件簡單的工作。文獻[12][16]說明了遊戲開發是橫跨了多個領域的系統整合，相較於一般軟體工程而言更加困難，而遊戲開發的生命週期也比一般軟體開發更為複雜，對初學者而言是一項嚴峻的挑戰。如何讓遊戲開發能夠更加快速且便利，更是在遊戲領域被廣泛探討的問題。

一種被廣泛提出的簡易開發機制是使遊戲開發變的更加直覺化。例如以敘事的方式進行遊戲的開發[7]，而敘事本身需要透過語言的描述，也因此許多遊戲引擎都會引進不同的腳本系統[8]，讓使用者透過腳本語言進行遊戲開發。文獻[6][11]也分別導入不同的腳本系統於遊戲開發中，降低開發難度。

另一種簡化開發的方式是改善遊戲開發的介面與工具。文獻[9][10]針對非程式專長的遊戲開發人員，設計專門的遊戲編輯工具，減少開發負擔。文獻[13]針對靜態的遊戲事件進行加值，導入動態參數使其更具變化性。文獻

myweb

[14]則提出一種抽象級別的轉換，使遊戲元件可以適用於不同類型的遊戲。文獻[15]則提出導入設計模式的概念於遊戲開發，來因應更複雜的行動平台遊戲開發。在 IS.GD 4.0 的遊戲開發介面當中，也提出了一種基於 Non-frame Based 的開發機制[3]，讓遊戲開發初學者可以在不了解遊戲引擎底層 frame 更新的狀態下，進行遊戲邏輯的編寫。

2.7. Leap Motion

Leap Montion[18]是一套用於獲取並辨識人類手掌動作的動作感應技術，透過紅外線 LED 以及接收器建立出約半平方公尺三維空間，在感應範圍內可以準確地抓取人類的雙手以及手勢，動作誤差小於一公分，每秒可以達到約 200 次的更新頻率。Leap Montion 提供開發者一套提取手掌資訊以及手勢辨識結果的 SDK，使用者可以透過 SDK 將 Leap Montion 與各項應用做結合。在 2.0 版本的 Leap Montion SDK 中，加入了精確的手部骨架追蹤，可以辨識如捏、抓等動作，同時增加了分辨左右手、五指以及手指關節等更細部的手掌資訊。也提供了新的 API 來確保數據的可信度，可以透過預測等方式模擬交疊手勢的下一步動作。

三、研究架構

3.1. 基礎架構概述

本研究架構以 Gameplay3D 為底層核心引擎，以該引擎為基礎向上架構更高階的使用者開發介面，參考 IS.GD 簡易開發架構，將其整合至原 Gameplay3D 遊戲引擎當中。本研究沿用 Gameplay3D 引擎繪圖、物理、音效以及跨平台管理等功能，再將各項功能依需求加以包裝、修改或新增，同時加入新的簡易開發架構如：使用者自定義遊戲狀態、抽象事件、Non-programing Based 群組物件操作以及人機操作辨識抽離架構。本研究將此重新架構後的引擎命名為 IS.GD 5.0，其系統架構如下：

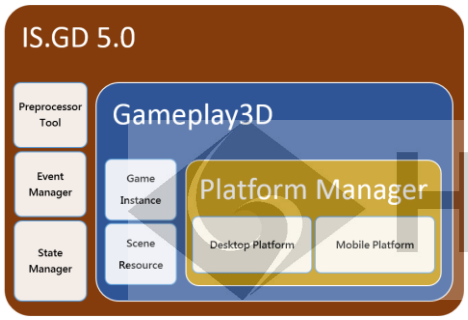

圖1、IS.GD 5.0 系統架構

本實驗室歷屆所開發之 IS.GD 引擎皆建構於 open source libraries 的開放資源[20]，而 IS.GD 5.0 引擎與前人所撰寫之 IS.GD 2.8 以及 IS.GD 4.0 引擎不同之處在於平台的相容性以及平台效能優化。由於 IS.GD 2.8 底層的 Ogre 繪圖函式庫過於龐大，不適合移植至行動平台上，在發展上受到限制。而 IS.GD 4.0 引擎則受限於 GFX 引擎於 Android 平台的優化不佳，執行效能遠不如 iOS 系統，相較而言 Gameplay3D 引擎的執行架構更為穩定，前景較優，也因此被本研究選用為引擎底層架構。

3.2. 遊戲引擎狀態管理機制

一般遊戲引擎常見的基礎狀態包含：Initialize、Update、Finalize，其中 Initialize 負責遊戲系統以及遊戲場景的初始化工作，完成之後便交由 Update 進行遊戲的更新迴圈，最後由 Finalize 進行記憶體釋放等遊戲終止工作。除了此三項基本的遊戲狀態，各遊戲引擎會依其功能需求或設計理念自行新增各式各樣的遊戲狀態，如 Gameplay3D 遊戲引擎將遊戲的 Update 狀態分為物件的狀態更新與繪圖的狀態更新，分別對應到 Update 與 Render 兩個狀態。更甚者可以再將遊戲的物理狀態、遊戲的網路狀態、遊戲的動畫狀態等加以細分，變成更多的狀態群組並加以管理。

上述的遊戲狀態其實包含了引擎運作本身的工作，對高階使用者而言並不需要去了解。然而在遊戲開發時，使用者會在各狀態插入遊戲本身需要的狀態邏輯，也因此引擎通常會提供給使用者一個撰寫狀態邏輯的介面或窗口。各遊戲引擎提供使用者進行狀態操作的介面也各不相同，以 IS.GD 2.8、IS.GD 4.0 以及 Gameplay3D 三款引擎為例，此三款引擎皆提供使用者固定的 C++狀態函式，使用者可於函式內部撰寫該狀態下需要進行的遊戲物件操作與遊戲邏輯。舉例來說使用者會在 Initialize 撰寫遊戲參數以及遊戲場景的初始化，在 Update 撰寫計時器功能或判斷遊戲結束的條件式，在使用者關閉遊戲的時候於 Finalize 狀態撰寫遊戲進度的保存等工作。

就以上三款引擎所提供的狀態管理機制，使用者可以撰寫遊戲邏輯於各個狀態，而不用理會引擎的運作。但對於遊戲本身所隱含的使用者自定狀態，則無法一一對應到上述引擎所提供的函式。就本校大學部進行遊戲專題製作的實務觀察發現，若使用者想實作自定義的遊戲狀態，只能藉由各種條件判斷式來檢測當前遊戲的狀態，這些判斷式就像一個又一個待檢測的旗標，不斷的在 Update 中輪詢，我們將其稱為 Flag。當 Flag 被檢測通過時，會

執行該 Flag 底下定義的相關工作，藉此模擬遊戲狀態的轉換。示意圖如下。

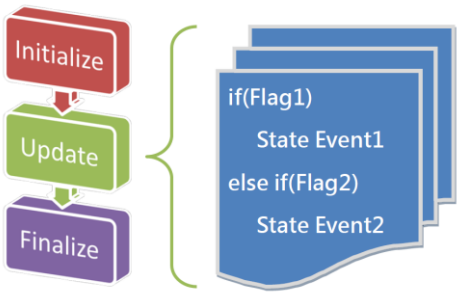

圖2、在 Update 中以 Flag 表示遊戲狀態

觀察此架構下的遊戲狀態管理可以發現，由於使用者定義的遊戲狀態無法對應到引擎所提供的狀態函式，導致使用者需要產生大量的 Flag，並將整個遊戲的遊戲狀態管理撰寫於 Update 狀態函式當中，使 Update 狀態函式下的巢狀階層不斷延伸、程式碼過度冗長且夾雜各種互無關聯的遊戲邏輯，隨著遊戲專案越做越龐大，程式本身會越來越難以閱讀，使用者還必須自行管理各個 Flag 所代表的遊戲狀態，不管是修改或者增加新功能，都相當的困難。由於所有遊戲相關狀態都撰寫於 Update 狀態函式，使用者在開發前還需要先了解引擎 frame based 的運行架構，對於初學者來說相當不直覺。

如前所述，本研究所撰寫之遊戲引擎是架構於原 Gameplay3D 遊戲引擎之上，也因此不可避免的繼承了 Gameplay3D 引擎的狀態管理架構，有鑑於該架構與前人所撰寫之遊戲引擎 IS.GD 2.8 與 IS.GD 4.0 相同，且同樣存在實務上進行遊戲狀態編輯的不便之處，本研究將針對此點進行遊戲狀態管理架構的改善，將於下一小節逐步說明。

3.2.1. 使用者自定義遊戲狀態

在改善遊戲狀態管理的研究中，有學者提出將遊戲可能遇到的狀態進行通用性的分類[8]，藉此達到更彈性的遊戲狀態管理，然而儘管增加了遊戲引擎所提供的狀態函式，仍無法滿足各類型遊戲開發時所需求的遊戲狀態。本研究提出一種可以讓使用者自定義遊戲狀態的管理機制，同時提供在不同情況下進行狀態轉換的轉換機制。對於使用者自定義的遊戲狀態而言，各狀態擁有各自的狀態內容供使用者編輯，狀態與狀態間彼此獨立，同時又存在狀態切換的關聯。

本研究沿用 IS.GD 2.8 與 IS.GD 4.0 遊戲引擎資料驅動層的概念，將資料的屬性設定與實體定義自遊戲程式碼中抽離。透過此概念，使用者在新增遊戲狀態時，只需要編輯遊戲狀態的文件即可，透過本研究撰寫的遊戲引擎前處理套件，會自動將使用者新增的狀態儲存於遊戲引擎的狀態列表當中，同時自動產生與該狀態相對應的狀態程式檔案，供使用者以程式語言撰寫該狀態下的遊戲邏輯。為減少使用者在撰寫狀態初始化時的工作量，本研究將部分常用的初始化功能加以包裝簡化，包含：場景初始化、遊戲背景音樂初始化、攝影機初始化、遊戲事件初始化、外部裝置初始化等常用功能，使用者可於遊戲狀態文件中撰寫相關屬性參數，引擎便會在遊戲進入該狀態時自動進行各項初始化工作，避免使用者撰寫重複且類似的工作。

在使用者定義完狀態文件後，所有被宣告的狀態都已經存在於狀態列表當中，但是要在各個狀態間進行切換而不產生衝突，則需要狀態轉換機制來管理，本研究提供的狀態轉換機制透過狀態堆疊的資料結構，延伸出狀態保存、狀態還原兩項功能。首先說明選擇以狀態堆疊的資料結構來儲存遊戲狀態的原因。在遊戲開發的實務上，為達到安全的遊戲物件記憶體管理，我們會套用下列方法。假設在某一遊戲狀態的 Initialize 底下創造了名為 A 的遊戲物件，通常會在同一狀態底下的 Finalize 函式將 A 物件刪除，如此一來可避免不必要的記憶體佔用，或是在 A 物件已經存在的情況下再次產生重複的物件，導致存取錯誤。然而就遊戲開發需求來說，可能存在著希望 A 物件保留至下一個遊戲狀態並繼續使用的情形。在此情況下，我們並不希望在狀態轉移的時候去執行 Finalize 函式，而能達到此功能的架構正是狀態堆疊的資料結構。

本研究提供的狀態轉換機制可以分為縱軸轉換與橫軸轉換，在同一狀態堆疊內的狀態切換為縱軸切換，反之將當前狀態堆疊結束並切換至另一狀態堆疊的轉換為橫軸轉換。在縱軸狀態轉換機制下本研究提供使用者 state_push 與 state_pop 這兩種轉換函式，橫軸轉換函式為 state_switch 並帶有額外的轉換參數。狀態堆疊結構與狀態轉換示意圖如下。

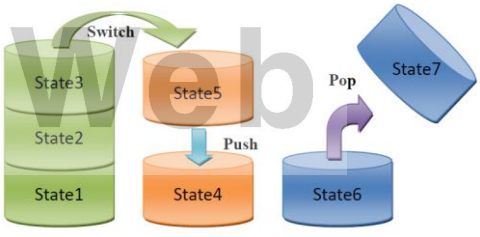

圖3、遊戲狀態堆疊與狀態轉換

接下來要逐一介紹狀態轉換函式 push、pop 以及 switch 的功能與限制，同時說明如何實現狀態保存、狀態還原兩項功能的狀態管理。

1.push：state_push 為最基礎的狀態轉換函式，該函式需要一個代表遊戲狀態名稱的參數 state_id，對應到狀態列表的某一個狀態。執行時將當前要切換的狀態置入狀態堆疊中，同時執行在狀態文件中宣告的初始化行為，並執行該狀態自身的 Initialize 函式。

2.pop：state_pop 是與 state_push 相對應的狀態轉換函式，將當前遊戲狀態倒回至前一個遊戲狀態，會執行當前狀態的 Finalize 函式。若進行 state_pop 時發現當前遊戲狀態與前一個遊戲狀態所使用的場景資訊不相同，將會在 Finalize 之後重新初始化前一狀態的場景資訊。

3.switch：state_switch 具有切換當前狀態堆疊的功能。state_switch 需要一個代表狀態名稱或狀態堆疊的參數 state_id，以及一個 rollback 的布林參數，代表是否要將轉移前的狀態堆疊進行 state_pop 至最後一個元素後在進行狀態切換，函式多載的 state_switch 還擁有一個選擇性的 protection 參數，此參數的用途在於實現狀態保存與狀態還原。當使用者進行 state_switch 的轉換動作時，可於 protection 參數替轉移前的狀態堆疊進行命名，便可將該狀態堆疊的狀態進行保存，同名稱的狀態堆疊會視為堆疊更新而覆蓋。如果使用者想要進行狀態還原的動作，只要再次呼叫 state_switch 的轉換函式並於狀態名稱參數 state_id 的部分輸入需要還原的狀態堆疊名稱即可。

在此架構下，switch 就像是一個橫向的狀態游標，用於標示當前運作的狀態堆疊或者新建立一個狀態堆疊。push 與 pop 則是在狀態堆疊中縱向移動的狀態指針，明確指出當前運行的遊戲狀態是哪一個。

對於狀態保存以及狀態還原這兩項管理機制，還有一些額外的說明與補充。本研究的狀態保存機制並不會完全備份狀態堆疊所產生的遊戲場景、遊戲物件等相關遊戲資源的實體，而是對狀態關聯以及堆疊階層的資訊進行保存，原因是要避免重複的物件實體造成龐大的記憶體開銷。在狀態還原時會依所儲存的狀態關聯以及堆疊階層，重新初始化至該狀態堆疊最後被保存的狀態，也就是說會重新的進行場景與遊戲物件的動態載入與初始化。

3.3. 遊戲事件管理機制

遊戲事件意旨任何在遊戲流程中可能觸發的動作與行為，基於引擎多元功能導向的開發概念，事件的類型往往相當繁雜。因應不同來源的輸入，在引擎中也會產生相對應的事件類別。然而在跨平台遊戲開發上，因各平台所擁有的輸入裝置不同，若要對應到不同的遊戲平台，即使遊戲內容與遊戲邏輯本身不用做更動，卻常常要撰寫多種平台下的輸入事件，仍有諸多不便。此節將敘述本研究在遊戲事件管理機制上所增強的功能及優化工作。

3.3.1. 遊戲事件編輯

在原生的 Gameplay3D 引擎上，其遊戲事件透過虛擬函式繼承的方式，讓使用者於各事件函式內撰寫該類別事件的程式邏輯。引擎本身透過名為 platform manager 的跨平台管理架構，在引擎初始化時自動進行該平台上各類輸入裝置的連接與初始化，並且在每個 frame update 去偵測各裝置是否有狀態變更。當檢測到輸入裝置有狀態變更，platform manager 會將接收到的裝置輸入透過函式呼叫的方式，執行相對應的事件。在函式內部使用者必須自行撰寫不同輸入值所對應的觸發行為，例如在 keyEvent 函式內定義當輸入值為鍵盤方向鍵→時，呼叫角色向右走的函式、當輸入值為鍵盤按鍵方向鍵←時，呼叫角色向左走的函式。

經由上述 Gameplay3D 引擎的事件觸發架構發現，在原生的 Gameplay3D 引擎上要撰寫遊戲事件都必須透過撰寫 C++程式語言來達成，且因應不同平台上的裝置輸入，需要分別撰寫各類裝置所對應的程式內容，並不存在一個較高階的事件編輯方式。對於遊戲事件的編輯，在 IS.GD 2.8 與 IS.GD 4.0 遊戲引擎皆提供 XML 資料格式的資料驅動層，包含了定義事件觸發的 Trigger.xml，以及連結事件觸發行為的 Proxy.xml。在 Trigger 文件中我們可以定義事件的名稱、裝置類型、輸入值、作用物件以及事件觸發頻率等事件屬性，在 Proxy 文件中我們指派當 Trigger 中宣告的事件被觸發後，遊戲需要去執行的動作或是函式。此種將資料屬性定義與程式碼抽離的架構，可以在不更動 C++程式碼的狀況下，對事件進行編輯，相較於 Gameplay3D 的遊戲事件撰寫，不僅更容易編寫，也更利於學習。也因此本研究引進了此種將資料屬性抽離的資料驅動概念，新增於 Gameplay3D 遊戲引擎當中，本研究規範了新的 Trigger 與 Proxy 文件格式，實作了包含 Trigger 文件與 Proxy 文件的解析與資料轉換、建立與 Gameplay3D 引擎事件機制的接口、以及將 Proxy 文件與 C++函式進行連結的

前處理套件，初步簡化遊戲事件編輯。

即便引進了 IS.GD 引擎的事件驅動架構，在編輯 Trigger 文件上，仍需要撰寫對應多種輸入裝置的 Trigger Event。由於 IS.GD 2.8 與 IS.GD 4.0 引擎並沒有橫跨 PC 與行動平台，輸入事件的類別較少，編輯上不會發生同一事件需要撰寫不同事件類別來因應跨平台的遊戲開發。Gameplay3D 因為同時支援了 PC 以及行動裝置平台的輸入裝置，直接套用舊有的 IS.GD 引擎架構將會有操作事件重複定義且 Trigger 文件過於複雜的問題，因此在規範新的 Trigger 文件格式的同時，本研究提出了一種事件抽象化的事件描述方法，於下一小節介紹。

3.3.2. 遊戲抽象事件

抽象事件的概念是透過分析不同類型的遊戲事件，找出各事件間相似的操作行為或操作目的，設計出新的事件通用屬性，達成單一事件描述就可以因應不同平台裝置的目的。除此之外，抽象事件還隱含了使用者進行操作時的命令意圖，透過此資訊可以讓事件在觸發的同時，呼叫引擎進行額外的物件處理，提前取得事件觸發時會使用到的資源物件，進一步簡少使用者撰寫遊戲事件的負擔。

本研究所提出的抽象事件如下，分別為：button_event、select_event 以及 rotate_event，以下將逐一介紹。

1.button_event：button_event 是遊戲中鍵盤事件的改良，而鍵盤事件是在 PC 遊戲開發時最常被使用、也最容易理解的遊戲事件。以使用者的操作行為來看，鍵盤事件即是希望在使用者按下某一按鍵的時候，觸發某個相對應的遊戲事件，例如：按下空白鍵時觸發名為 Jump 的 Action。多數的行動平台裝置本身並不搭載實體的鍵盤，即便有像是虛擬鍵盤的系統內建功能，卻很難應用於遊戲操作上。因此當我們想要在行動遊戲上達成按下按鍵後觸發按鍵事件的行為，通常會在遊戲畫面上加上虛擬的 button，藉由觸碰螢幕上的 button 來觸發按鍵事件。本研究所使用之 Gameplay3D 遊戲引擎本身也提供了此類型的虛擬按鍵功能，但如前一小節所述，對於所有 button 按鍵的觸發，使用者都必須於特定函式內撰寫，且與鍵盤事件的函式各不相關，同一功能卻需要再兩個不同的函式重複撰寫。在簡化成 Trigger 事件定義的架構後，我們將此二項事件的定義方式整合，稱為 button_event。

2.select_event：此事件顧名思義隱含了使用者進行選擇此一動作的意涵。在遊戲中，若使用者點選了螢幕上的一個遊戲物件，往往是想要對該物件進行操作，或是獲取該物件的相關訊息。從輸入偵測的行為來看，在 PC 平台上我們會以滑鼠游標以及滑鼠按鍵進行選擇的動作，在行動平台則是以手指為游標，直接觸碰屏幕上顯示的物件進行選擇。除此之外，各式體感裝置也有所謂的虛擬游標以及代表選取的動作辨識。選取事件除了將上述三種不同裝置的輸入行為整合外，對於被選取物件，本研究也進行了簡易開發的處理。由於使用者是透過屏幕的投影畫面來看到遊戲的三維世界，選取物件的時候實際上是將畫面上被選取的二維座標，以數學式與物理運算轉換成遊戲中三維座標，通常是以物理引擎的 ray 功能來進行實作。在原先的滑鼠或是觸碰事件中，只會傳遞引擎偵測到的二維座標輸入供使用者應用，包含上述的數學式以及物理運算使用者必須自行加工，即便在前人的研究[3], [4]上已經將 ray 的常用物理功能加以包裝簡化，但對於使用者來說仍要先了解 ray 的運作原理，學習上相當不易。本研究架構中的 select_event 本身隱含了使用者進行選取的操作行為，因此在事件觸發時會自動將被選取物件傳遞給使用者使用，達到簡易開發的目的。

3.rotate_event：由於行動平台設備大多搭載了陀螺儀與加速度計等內置硬體設備，遊戲開發上也常常看到以傾斜、旋轉設備等方式來進行操作。在 PC 平台上雖然沒有可以達到同樣功能的基礎設備，卻因為近年來體感裝置的盛行讓 PC 平台上出現了眾多的外接式硬體裝置。rotate_event 即是將行動裝置上的旋轉操作與體感設備抓取到的旋轉動作整合，並且以簡單的 Trigger 事件定義來完成，以 Leap Motion 為例就是將人類手掌的翻轉量與傾斜量對應到行動裝置的陀螺儀數據，藉此達到類似的遊戲體驗。

本研究除了定義出以上三種抽象事件外，在遊戲事件的操作上，尤其是 select_event 的應用上還定義了一種群組物件篩選與操作的機制，可以達成選取特定類別物件或是對大量群組物件同時進行操作的目的，將在下一節介紹。

3.3.3. Non-programing Based 群組物件操作

在一個遊戲事件發生的同時，為了方便存取與該事件相關的遊戲物件，通常會有一個方便開發者進行操作的事件目標被傳遞，我們稱之為 Event Target。以鍵盤事件控制遊戲角色移動的例子來說，按下鍵盤按鍵的動作為事件觸發，遊戲角色本身為被傳遞的 Event Target，而角色產生位移即為事件觸發後的動作。此架

Airiti

構本身是以方便開發者存取物件的角度去設計，在 IS.GD 2.8 遊戲引擎中便已經被實做出來。但在該版本引擎中，遊戲事件與 Event Target 的關係被設定為一對一，即是說當一個事件被觸發的同時，只會有一個 Event Target 被傳遞。若要對大量的群組物件進行操作，則需要撰寫大量的遊戲事件，分別對應到不同的 Event Target，或者直接在 C++程式碼中以搜尋場景物件的函式 find_node_in_map 來索引到各個需要進行操作的物件，並以迴圈的方式逐一處理。

針對此種單一事件對應單一物件的架構缺陷，也有前人進行相關研究並加以改善，文獻[2]針對此問題提出了一個解決方案。該方案透過宣告遊戲狀態模板以及作用對象，在 XML 撰寫程式邏輯上的 for 迴圈，便可以避免定義大量且相似的遊戲事件，但此方法類似於將原先 C++的程式碼精簡過後移至 XML 資料定義層，對於遊戲事件的簡易開發來說效果有限。文獻[3]則提出了一種基於 Non-frame Based 概念的事件宣告方式，該方法以物件陣列的方式傳遞 Event Target，並且在 XML 將事件 Type 設定為 for 屬性，即可達到對物件陣列內所有元素進行群組操作的目的。上述兩種方法在簡易遊戲事件開發上皆有貢獻，但仍有部分問題仍未被解決。首先兩種方法對於群組物件的定義，基本局限於屬性相似或者本身為同一物件經多重產生後的物件陣列。對於不同類別的遊戲物件，並不會同時存在於同一物件陣列。舉例來說當遊戲中存在著 A 類別與 B 類別兩種 NPC 怪物，兩個類別的怪物群組分別有不同的 AI 與各自的物件參數。在遊戲中為方便管理，我們會將 A 類別的物件與 B 類別的物件分開存放於各自的物件陣列。但如果今天要新增一個 A 類別與 B 類別皆會被影響的遊戲事件，就上述兩種方法仍需要撰寫兩個 for 迴圈邏輯的事件，來分別對 A 類別與 B 類別的物件群組做處理。此外上述兩種改善方法基本上仍帶有以 for 迴圈程式邏輯為基礎的操作方式，在使用上需要先行帶入程式邏輯的執行概念才可以進行群組物件的操作。

對於群組物件的操作，本研究透過增加物件 Tag 的方式，讓使用者在 Gameplay3D 引擎編輯場景物件的.scene 檔案下對各物件進行群組分類，單一物件可以同時擁有多個物件群組的 Tag，藉此對應到各種通用的物件特性。

圖4、單一物件擁有多個群組標籤

在進行遊戲事件編輯的時候，只要在原先傳遞單一物件的 Event Target 欄位填入"tag :tag_name" 的格式，就代表該事件觸發時是對該 Tag 群組的所有物件進行處理。如此一來便可以實現單一事件針對不同物件群組同時進行操作的一致性開發，且使用者在撰寫該事件時並不需要帶入 for 迴圈的程式邏輯，達到 Non-programing Based 的物件群組操作。

除了可以同時對多個物件群組進行操作外，將此概念與前一小節的 select_event 相結合，就可以達到在 select_event 時以 Tag 篩選可以被作用的 Event Target 群組，例如在 select_event 的篩選項填入名為 NPC 群組的 Tag，就代表此事件只有在選取到屬於 NPC 群組物件的時候，才會進行事件的觸發。

3.3.4. 人機操作辨識抽離架構

隨著科技的進步，應用於遊戲操作的硬體裝置日新月異，尤其是用於人類肢體動作辨識或者手勢辨識的體感裝置於近年來大幅盛行，各遊戲引擎對於外部裝置的連接與開發都有各自的支持。本研究也實作了一種基於人機操作辨識抽離的外部裝置連接架構於 Gameplay3D 引擎中。對於應用外部裝置在遊戲開發的實務上，透過觀察本校大學部遊戲專題製作的使用者行為發現，使用者常常會將操作辨識與遊戲邏輯撰寫在同一區塊的程式碼中，形成操作辨識與遊戲邏輯具有相依性的情況。造成程式碼複雜度增加，可讀性降低，一旦要進行程式改寫動作時，將難以進行維護。

為避免使用者在撰寫操作辨識與遊戲邏輯時產生程式相依的情形，在外部裝置的開發上，套用本研究在前面章節敘述過的遊戲事件編寫機制，並透過操作辨識、事件觸發、執行事件動作的流程架構，將操作辨識與遊戲邏輯各自獨立，架構運作如下圖。

圖5、操作辨識獨立架構

四、實務應用

4.1. 使用者自定義遊戲狀態

本章節實際展示使用者自定義狀態的宣告方式與各項屬性參數，以及遊戲狀態如何對應到遊戲事件的管理。

首先展示在 UserDefineState.state 文件中定義狀態以及設定狀態屬性的寫法：

```
state MenuState
{
        scene = Scene1
        background_sound = Mabinogi.ogg
        camera
        {
                follow = god
                look = god
                location = 0,10,0
                distance = 50
                yaw = 0
                pitch = 0
                orientation_follow = true
        }
        select_event enable
        {
                pick
        }
        button_event enable
        {
                CamaraControl_W
                CamaraControl_S
                CamaraControl_A
                CamaraControl_D
        }
}
```

圖6、使用者自定義狀態

- state：代表一個遊戲狀態的宣告，後方參數為使用者自定義狀態的名稱。
- scene：進入該狀態時希望進行初始化的場景名稱。
- background_sound：在該狀態底下時希望自動播放的背景音樂。
- camera：設定攝影機的相關參數。follow：攝影機跟隨目標的名稱，可為空值。look：表示攝影機看向的物件或三維座標。location：非跟隨狀態時攝影機在場景中的絕對位置。distance：設定攝影機與被跟隨物件的距離，可為空值。yaw：攝影機 Y 軸的偏移角度。pitch：攝影機的仰角。orientation_follow：是否要與跟隨目標的面向同步。
- select_event：是否要啟用 select_event，並且設定要啟用的 event。
- button_event：是否要啟用 button_event，並且設定要啟用的 event。

定義完上述遊戲狀態後，透過本研究的前處理程式，引擎將會自動產生與該遊戲狀態相對應的 C++程式文件，新產生的空文件包含三個基本狀態函式，該狀態函式從遊戲引擎的狀態工作中，獨立出來的子狀態函式，方便使用者撰寫只與該狀態相關的遊戲行為。

4.2. 遊戲抽象事件

本小節將實際展示抽象事件的宣告方式與各項屬性參數，並且說明如何管理類別群組物件。

以下為 select_event 的事件宣告與相關參數，撰寫於 UserTrigger 文件當中。

```
select_event  PickScale
{
        event_name  pick
        {
                type = press
                interval = 1
                action = BoxScale
                target = select
                select = tag(box_group)
        }
}
```

圖7、select_event 事件宣告

- select_event：代表一個 select_event 群組的宣告，群組名稱為 PickScale，一個 event 群組裡面可以包含多個 event，群組名稱可為空值。
- event_name：代表事件名稱，在所有 select_event 中必須是唯一。
- type：表示此事件的觸發條件，press 代表點擊滑鼠或是觸碰螢幕的瞬間、move 代表滑鼠或手勢的拖曳、release 代表釋放滑鼠按鍵或是手指離開觸碰螢幕的瞬間。
- interval：此事件的最高觸發頻率，以毫秒為單位，最快每毫秒可觸發一次。

HyWeb

- action：此事件偵測到觸發時要執行的 Action。
- target：為事件觸發時會被作為引數傳遞的 Event Target，可以為單一物件名稱或是物件群組名稱，若寫作 select 則會依選擇事件所選擇到的物件進行引數傳遞。
- select：限定會被選取的物件範圍，可以為單一物件名稱或是物件群組名稱，是一個物件選取的篩選項目。

此 select_event 的範例實作了一個當選擇到 box_group 群組的物件時，會對該物件進行 BoxScale 動作的簡單定義。觸發動作的相關設定會撰寫於 UserProxy 文件當中，其樣貌如下：

```
action_name  BoxScale
{
        scale = 1.2,1.2,1.2
}
```

圖8、名為 BoxScale 的 Action 行為定義

- action_name：與 Trigger Event 中的 action 屬性相對應，藉此連結事件定義與觸發後行為定義。
- scale：實作於引擎 Proxy Action 架構下的一個 inline 函式，直接對傳入的 Event Target 進行操作，此處會對傳入物件進行縮放的動作。

經由上述的 Trigger Event 事件宣告與 Proxy Action 的動作行為定義，在遊戲中的可以達到的實際功能是，當選使用者進行點擊屏幕的操作時，若選取到屬於 box_group 群組中的其中一個物件，會對該物件進行放大 1.2 倍的行為。此事件同時作用於滑鼠事件與觸碰事件，使用上不必重複宣告。接下來將介紹 button_event 的宣告方式與相關屬性：

```
button_event  Characters
{
        event_name  jump
        {
                type = press
                key = KEY_C
                button = Jump
                interval = 1
                target = Andys
                action = CharacterJump
        }
}
```

圖9、button_event 事件宣告

- button _event：代表一個 button _event 群組的宣告，群組名稱為 Characters，一個 event 群組裡面可以包含多個 event，群組名稱可以為空值。
- type：表示此事件的觸發條件，press 代表按下鍵盤按鍵或是觸碰 button、hold 代表長壓鍵盤按鍵或者 button、release 代表釋放鍵盤按鍵或是手指離開 button。
- key：代表鍵盤輸入的按鍵，與 button 至少要有一項有設定值。
- button：代表虛擬按鈕的名稱，與 key 至少要有一項有設定值。

上述 button _event 的事件表示，當按下名稱為 Jump 的按鈕或是鍵盤按鍵 C 的時候，會對名為 Andys 的遊戲物件執行 CharacterJump 的動作行為。最後展示本研究架構 rotate_event 的宣告方式：

```
rotate_event
{
        event  BoxRotate
        {
                roll = 30,35
                pitch = non
                interval = 1
                target = Box
                action = BoxRotation
        }
}
```

圖10、rotate_event 事件宣告

- rotate_event：代表一個 rotate_event 群組的宣告，群組名稱為空值。
- roll：繞 Z 軸旋轉的事件的觸發條件，由兩個傾斜角度構成，當傾斜介於兩角度之間時事件觸發，可以為空值。
- pitch：繞 X 軸旋轉的事件的觸發條件，由兩個傾斜角度構成，當傾斜介於兩角度之間時事件觸發，可以為空值。

上述 rotate_event 實作了一個簡單的事件，當設備旋轉角度介於 30 跟 35 之間時，會對 Box 進行旋轉的動作。可作用於陀螺儀感應與桌上平台之 Leap Montion 等外接裝置。

除了本節所介紹的三種抽象事件外，本研究也實作了其他遊戲事件於 IS.GD 5.0 之上，包含：時間事件、物理碰撞事件、外部裝置事件如 LeapEvent 與 KinectEvent 等。但由於此

類一般通用性的遊戲事件皆為前人學者於過去 IS.GD 引擎研究時，就已經提出並且實作，並非本研究貢獻，在此就不多做相關介紹，只說明 IS.GD 5.0 具備過去 IS.GD 引擎所擁有的遊戲事件與其功能。

五、結論

本研究提出一種供使用者編輯遊戲流程狀態的引擎架構，狀態間彼此獨立，並透過轉換函式來進行切換。透過狀態堆疊的資料結構，使用者可以快速的保存或還原各階段的遊戲狀態。狀態本身同時可以管理遊戲事件的啟用與否，減少單一事件個別管理的負擔。因應跨平台引擎在各平台下的事件輸入，提出一種基於抽象概念的一致性事件宣告。解決事件重複定義的開發負擔，並且將事件隱含的命令操作提前完成，簡化引擎事件觸發架構下的流程。對於人機操作辨識與遊戲邏輯的相依性，提出一個開發框架，將操作辨識與遊戲邏輯各自獨立，並透過事件觸發的機制來呼叫，使遊戲程式更容易進行維護。整合並改善前人所提出之引擎架構，並實作於 Gameplay3d 引擎之上，同時補強該引擎所欠缺之高階使用者介面，使其更易於使用於遊戲開發。

參考文獻

[1] 鄭武堯、陳志傑、郭聰儒，“應用設計樣版及 XML 文件於3D遊戲引擎架構設計，”第十四屆物件導向技術及應用研討會, 2003.

[2] 陳詩愷，“多重開發網路平台與簡化遊戲架構設計實作，” 世新大學資訊管理學系碩士學位論文, 2011.

[3] 吳昱翰，“簡易開發暨載入階段優化之 3D 遊戲引擎架構，” 世新大學資訊管理學系碩士學位論文, 2012.

[4] 余家瑄，“跨平台遊戲引擎物理與網路之簡易開發，” 世新大學資訊管理學系碩士學位論文, 2013.

[5] G. G. Koch, M. A. Tariq, B. Koldehofe, and K. Rothermel, “Event processing for large-scale distributed games,” Proc. Fourth ACM Int. Conf. Distrib. Event-Based Syst. - DEBS ’10, p. 103, 2010.

[6] I. Suzuki, K. Tsunoda, and R. Hishiyama, “Game description language and frameworks for langrid gaming,” Proc. 29th ACM Int. Conf. Des. Commun. - SIGDOC ’11, p. 67, 2011.

[7] H. Wei, “Embedded narrative in game design,” Proc. Int. Acad. Conf. Futur. Game Des. Technol. - Futur. ’10, p. 247, 2010.

[8] W. White and C. Koch, “Better Scripts , Better Games,” 2007.

[9] H. Yoo and S. Kim, “EDGE : An Easy Design Tool of Game Event for Rapid Game Development,” no. 1, pp. 2–3, 2010.

[10] O. H. Cho and W. H. Lee, “An interactive event-design tool for rapid game development,” in Proceedings of the International Symposium on Consumer Electronics, ISCE, 2011, pp. 450–453.

[11] C. Yu, Q. Zhao, M. Chen, and Q. Liu, “Design and Realization of the Script System in an Educational Game Engine,” 2009 Second Int. Symp. Knowl. Acquis. Model., pp. 25–28, 2009.

[12] C. M. Kanode and H. M. Haddad, “Software engineering challenges in game development,” in ITNG 2009 - 6th International Conference on Information Technology: New Generations, 2009, pp. 260–265.

[13] J. Snowdon and A. Oikonomou, “Creating more entertaining and re-playable games by dynamically introducing and manipulating, static gameplay elements and events,” in Proceedings of CGAMES’2011 USA - 16th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games, 2011, pp. 94–100.

[14] V. T. Sarinho and A. L. Apolinário, “A generative programming approach for game development,” in SBGAMES2009 - 8th Brazilian Symposium on Games and Digital Entertainment, 2009, pp. 83–92.

[15] A. G. Peker and T. Can, “A design goal and design pattern based approach for development of game engines for mobile platforms,” in Proceedings of CGAMES’2011 USA - 16th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational and Serious Games, 2011, pp. 114–120.

[16] R. Ramadan and Y. Widyani, “Game development life cycle guidelines,” in Advanced Computer Science and Information Systems (ICACSIS), 2013 International Conference on, 2013, pp. 95 – 100.

[17] P. E. Black, “finite state machine,” Dict. Algorithms Data Struct., vol. 2007, pp. 1–7, 2008.

[18] “LeapMotion.” [Online]. Available: https://www.leapmotion.com/.

[19] “Gameplay3D.” [Online]. Available: http://gameplay3d.org/.

HyWeb