

國立臺北教育大學理學院數位科技設計學系玩具與遊戲設計碩士班

碩士論文

Department of Digital Technology Design – Master Program in Toy and Game

Design College of Science

National Taipei University of Education

Master's Thesis

以 L-System 為基礎之平台遊戲關卡生成之研究

A study of L-system-based Level Generation for 2D Platform
Game

何思頡

Sy-Jye Her

指導教授：王學武博士

Advisor : Hsueh-Wu Wang, Ph. D.

中華民國 104 年 6 月

June, 2015

謝誌

終於即將畢業踏上新的人生歷程，這一路上獲得許多人的幫助。首要我要感謝我的指導老師王學武教授這段日子以來的提攜與栽培，感謝老師給予我各方面的幫助，除了學業上的指導、指點迷津，拉拔我參加國科會等計畫，還有協助我處理其他各種瑣碎事務，我很感謝能有這個緣分在求學過程中能遇到像老師這麼好的老師。再來，我還要感謝范丙林教授與張耀勳教授，對我提供寶貴的意見，讓我能順利完成此篇論文。

摘要

本研究針對 2D 平台遊戲提出一個自動關卡生成的演算法，這個演算法是以 L-System 為基礎，並以關卡設計模型組織關卡內部架構與元件，提供一套可讓使用者自行定義個人化關卡模板且由系統程序性完成整個關卡生成的機制。

本研究參考地底尋寶與 Smith(2008)的關卡模型架構訂定關卡內部物件的邏輯關係，定義每個關卡為一個由 15 種樣式房間構成的棋盤，並以 L-System 疊代產生一連串連續的空間物件集合，搭配使用者設定的關卡物件和關卡難易度與邊界容錯處理實作出具備可玩性的遊戲關卡。

本研究的關卡生成演算法，可確保地圖上所產生的關卡包含必要的解決路徑以及提供大量隨機多樣的關卡生成結果，且藉由 L-System 的生長規則可輕易擴展出龐大、幾乎不重複的遊戲世界，再搭配可人為調整的參數控制，讓使用者擁有關卡內部物件的彈性調整。利用此生成機制來設計關卡能有效減少開發時間，且模組化後的關卡有利於整體調整遊戲內部元件和難易度，方便遊戲設計師掌握目前的遊戲架構。

關鍵字：平台遊戲、關卡設計、程序性生成、L-system

目錄

謝誌.....	2
摘要.....	i
目錄.....	iii
表目錄.....	v
圖目錄.....	vi
第一章 緒論.....	1
第一節 研究背景與動機.....	1
第二節 研究目的與研究問題.....	3
第二章 文獻回顧.....	4
第一節 平台遊戲(Platformers).....	4
一、遊戲類型(Videogame Genres)	4
二、平台遊戲(Platform Game).....	5
第二節 關卡生成(Level Generation)	7
一、程序性生成(Procedural Generation)	7
二、程序性內容生成遊戲(PCG-games).....	11
第三節 關卡與遊戲設計(Design Pattern of Game Level).....	26
一、關卡設計(Level Design).....	26
二、平台遊戲關卡設計框架(Framework).....	27
第四節 L-system.....	32
第五節 小節	36
第三章 研究方法.....	38
第一節 生成方法.....	38
第二節 邊界容錯.....	42
第三節 難易度處理.....	46
第四章 實驗結果.....	55
第一節 實驗定義.....	55
第二節 生成範例.....	59

第三節 參數變化.....	62
第五章 結論與建議.....	80
第一節 結論.....	80
第二節 後續建議與未來工作.....	82
參考文獻.....	84

表目錄

表 2-1 憤怒鳥關卡生成的各種類物件生成機率參數表	21
表 2-2 憤怒鳥關卡生成的各欄位物件數量機率參數表	21
表 2-3 憤怒鳥關卡生成的間距機率參數表	21
表 3-1 關卡生成的遞迴規則	39
表 3-2 關卡生成規則中的概念表	40
表 3-3 平台間距的生成機率表	50
表 3-4 敵人生成數量與難易度對應表	51
表 3-5 陷阱生成數量與難易度對應表	53
表 4-1 房間內的關卡設計元件	56
表 4-2 生成台階的路線規範表	65
表 4-3 敵人預定生成區域規範表	70
表 4-4 陷阱預定生成區域規範表	75

圖目錄

圖 2-1 Compton 與 Mateas 的遊戲關卡階層架構模型	10
圖 2-2 遊戲設計的區域結構	10
圖 2-3 Martin 的腳本模型架構	11
圖 2-4 地底尋寶的地圖生成範例	13
圖 2-5 地底尋寶的起始房間生成範例	14
圖 2-6 地底尋寶的路徑生成範例	15
圖 2-7 Jack Benoit 遊戲關卡生成範例	16
圖 2-8 地底尋寶的地圖填補範例	17
圖 2-9 地底尋寶的蛇窟生成範例	17
圖 2-10 template 與 obstacle blocks 範例	18
圖 2-11 地底尋寶的同樣式地圖生成比較範例	18
圖 2-12 憤怒鳥的關卡生成物件表	19
圖 2-13 憤怒鳥的關卡生成範例圖	20
圖 2-14 馬可夫鏈(markov chain).....	23
圖 2-15 Breadth-First Search 與 Dijkstra's algorithm.....	24
圖 2-16 範例 The Mario Map Representation	24
圖 2-17 程序化關卡生成案例	25
圖 2-18 瑪利歐 3 關卡內的平台實例	28
圖 2-19 瑪利歐 3 關卡內的障礙實例	28
圖 2-20 瑪利歐 3 關卡內的移動輔助器實例	29
圖 2-21 瑪利歐 3 關卡內的收集品實例	29
圖 2-22 瑪利歐 3 關卡內的觸發器實例	30
圖 2-23 Smith 的關卡架構概念模型	31
圖 2-24 Algae 範例	33
圖 2-25 Parametric DOL system 植物生成範例	33
圖 2-26 Sierpinski triangle	35
圖 2-27 Grammar Representation 的簡單範例	35
圖 3-1 架構概念圖	38
圖 3-2 系統流程圖	39
圖 3-3 14 種房間樣式	42
圖 3-4 關卡地圖預定空間	43
圖 3-5 根據預設值與使用者設定的交集決定實際棋盤大小	44
圖 3-6 使用者錯誤設定在邊界外的範例	45
圖 3-7 使用者錯誤設定在挖空區域內的範例	45
圖 3-8 使用者錯誤設定的孤島範例	45

圖 3- 9T 字型房間範例圖	47
圖 3- 10 額外產生台階示意	49
圖 3- 11 台階難易度 2 的生成示意圖	50
圖 3- 12 敵人物件生成步驟	52
圖 3- 13 敵人物件重疊處理	52
圖 4- 1 基本疊代的關卡生成範例	60
圖 4- 2 不規則外型的關卡生成範例	60
圖 4- 3 多重出口的關卡生成範例	61
圖 4- 4 相同自定義參數產生的不同關卡生成比較範例	62
圖 4- 5 範例遊戲中角色示意圖	62
圖 4- 6 房間內部背景與固定物件的生成範例	63
圖 4- 7 房間內部台階生成路線規劃範例	64
圖 4- 8 難易度 0 的台階生成範例	65
圖 4- 9 難易度 1 的台階生成範例	66
圖 4- 10 難易度 2 的台階生成範例	67
圖 4- 11 難易度 3 的台階生成範例	67
圖 4- 12 難易度 4 的台階生成範例	68
圖 4- 13 房間內部敵人預定生成區域規劃範例	70
圖 4- 14 難易度 0 的敵人生成範例	71
圖 4- 15 難易度 1 的敵人生成範例	71
圖 4- 16 難易度 2 的敵人生成範例	72
圖 4- 17 難易度 3 的敵人生成範例	72
圖 4- 18 難易度 3 的敵人生成範例	73
圖 4- 19 房間內部陷阱預定生成區域規劃範例	74
圖 4- 20 難易度 0 的陷阱生成範例	76
圖 4- 21 難易度 1 的陷阱生成範例	76
圖 4- 22 難易度 2 的陷阱生成範例	77
圖 4- 23 難易度 3 的陷阱生成範例	77
圖 4- 24 難易度 4 的陷阱生成範例	78
圖 4- 25 房間內部物件的生成範例	79

第一章 緒論

第一節 研究背景與動機

電子遊戲的起源可以追溯到 50 年代，在 70 年代開始以商業娛樂型態席捲媒體市場，隨著 1972 年 Atari 的成立，以兵(Pong)為首的街機遊戲(Arcade game)以及其他遊戲主機與家用電腦遊戲開始大量普及，並逐步滲透到大眾的日常生活，成為現代娛樂文化的主流產業之一。電子遊戲目前已經是一個成熟而且持續在成長產業，但隨著遊戲開發成本與競爭對手的增加，遊戲開發已經是一個高度競爭與高風險的行業。

伴隨產業競爭與成本考量，遊戲開發上對耐玩度(re-playability)的設計越來越重視。一款遊戲是否有趣？是否有趣到經得起一玩再玩，玩家不會因為玩過一遍就輕易的關掉遊戲，結束這款遊戲短暫的一生。然而延伸遊戲的壽命最直接的做法便是提升整體遊戲品質，但相對高品質需投入高成本。

近年來，許多遊戲即努力在原先的基礎上實踐、創造遊戲重玩的價值，如現今的線上遊戲通常透過多人模式、探索寬廣的世界或加入 RPG 元素中非線性的人物劇情。然而這些方法也並非沒有缺點，當遊戲世界不斷擴大、非線性的故事劇本不停延伸、多人機制帶來越來越多的玩家時，各種開發維護成本也同步攀升，如何在這之間找到個平衡點，便是一個重要的課題。

程序性內容生成(Procedural Content Generation)即是一個為了增加遊戲重玩的價值同時降低成本，普遍使用在遊戲中的作法。一種基於解決開發成本與創造更多遊戲經驗的平衡問題，以演算生成遊戲內容物的計算策略取代人工手動創作的形式。在擁有探索型地域的動作或冒險 RPG 的關卡設計中常見用來製作層層推進的地牢結構(dungeon)，讓玩家享受探索的樂趣。

雖然在視覺美感上，由程序生成的幾何圖案實在無法與人工繪製的畫面相互競爭，但它能有效率地賦予遊戲新的重玩價值。依據遊戲的性質與不同賣點，玩

家在一款遊戲追求的未必是精緻的視覺享受，而是有趣的遊戲歷程。實際上，大多數程序內容生成的遊戲，相對於單獨選用系統自動生成，較常採用結合人工繪製的元件與程序生成的做法，或是提供在系統生成後可以進行微調，使遊戲畫面與關卡走向可以更精緻和順暢。

程序性內容生成的遊戲發展已有段歷史，在近年隨著遊戲產業擴大、需求量攀升，程序性內容生成遊戲再次成為熱門焦點。在針對程序性內容生成遊戲的研究上，以 L-system 做為程序性關卡生成的研究有限，且多以動畫或背景中造景，如樹木、建築物為主，較少運用在關卡程序性生成的設計上，且關卡設計本身是具針對性的研究領域，不同遊戲類型的關卡生成模型在其他不同遊戲類型上的通用度普遍偏低。本研究將以此為出發點，希望運用 L-system 生成針對 2D 平台遊戲的關卡內容。

我們不討論如何生成一款好玩的遊戲，這個部分除了程序上的設計也包含了一些企劃、美術的設計層面，我們把問題替換成從程序的角度上如何能產生出不重複又能玩的關卡。在一些遊戲關卡設計上，如地底尋寶、暗黑破壞神與時空幻境等遊戲，存在各種讓玩家角色探索的地形區域或地圖，這些關卡是否能玩，常用是否存在路徑解答作為指標，這代表著遊戲關卡確實能被破關，遊戲才具有可玩性。為此，在設計程序生成的演算上必須確保在具有路徑解答的狀況下進行生成，否則就要在生成後反覆驗證。

而在考慮重玩性的要求上，一款遊戲所擁有的重玩價值難以被量化評估，但從過往程序內容生成遊戲的研究上可推斷，這些促使玩家一玩再玩的遊戲，有個很重要的設計在於遊戲經歷的豐富多樣性，這也類似於線上遊戲常會定期推出新活動以維持玩家對遊戲的熱情與新鮮感；玩家在重玩遊戲時仍能對遊戲抱有期待，預期會得到跟上一次不一樣的遊戲經歷與樂趣。

對此，在程序性生成的演算上須確保有大量的可能結果，而 L-system 本身的特性可使得這個想法實現，並把生成結果控制在一個邏輯範圍內；就結果而言能

適當管理關卡大小並使得關卡的生成程序具有邏輯規範，不至於因為隨機變數造成關卡平衡上的破壞，因此針對遊戲關卡的可玩性與再玩性，L-system 裡的產生式規則訂立將會非常重要。

我們可以期望在都市規劃(Parish & Müller, 2001)已有段發展歷史的 L-system 在遊戲關卡生成設計上依然可行，且近年的程序內容生成遊戲中已有以 L-system 為基礎設計關卡情境與場景之研究並以此產生的軍事訓練遊戲(Martin, Schatz, Hughes & Nicholson, 2010)，我們可以預期運用在平台遊戲上也能有良好的成果。

在 L-system 的框架之下，設計師可以發揮的地方主要在於規則的訂定與參數控制。因為 L-system 本身具有遞迴生長的特性，我們可以用遞迴次數限制住結果範圍，在當遊戲從小規模的關卡擴展到大型世界關卡時，可直接接續原本的規則讓關卡自行成長，省下一些遊戲開發成本。

第二節 研究目的與研究問題

壹、研究目的

本研究目的為運用 L-system 生成具可玩性與再玩性的程序性內容生成 2D 平台遊戲關卡。研究目的如下：

- 一、設計並提出以 L-system 為基礎的 2D 平台遊戲的自動生成演算法。
- 二、探討自動生成演算法與人為關卡鋪成的整合方式。

貳、研究問題

根據上述的研究目的，本研究提出以下的研究問題？

- 一、以 L-system 為基礎的 2D 平台遊戲關卡自動生成的機制為何？
- 二、人為控制的加入對關卡生成的影響為何？此外，使用者可控制、發揮的地方有哪些？

第二章 文獻回顧

本章共分為四個小節，第一節為了將平台遊戲的特色與其他遊戲類型作出區別，將從遊戲分類開始往下探討，說明遊戲類型不同於以往電影以劇情內容、故事背景，而以遊戲風格、機制、挑戰作為分類依據，最後整理出平台遊戲的基本元素。第二節為關卡生成之技術理論，將回顧程序性內容生成技術的歷史與簡介部分關卡生成研究中的模型架構，以及挑出幾款程序性內容生成遊戲作為範例、說明其中的生成原理。第三節為關卡與遊戲設計理論，在文獻回顧上將整理關卡設計與遊戲設計之間的相異處，以釐清本研究是針對功能導向的關卡設計研究，並列舉本文參考與沿用的關卡設計模型。第四節為 L-system 論述，初步簡介 L-system 的歷史、特性與文法定義，最後輔以實際的例子作為說明。

第一節 平台遊戲(Platformers)

本節將藉由討論遊戲的分類方式與歷史，闡述平台遊戲的特色與過往經典，以進一步濃縮出該遊戲類型的重要元素。

一、遊戲類型(Videogame Genres)

在遊戲分類上，遊戲可依據「主機平台」(Platform)、「類型」(Genre)、「遊玩模式」(Mode)以及「環境」(Milieu)這四層面向做不同分類(King & Kryzwinska, 2002)。其中「類型」是業界最常使用的分類依據，也是這四層分類中無統一規範而最特殊的一環；King 和 Kryzwinska(2002)介紹「類型」是針對遊戲本質上的一種特殊分類方式，同時坦言「類型」此一分類界定是來自更廣義的遊戲集合(the wider gaming community)。簡單說，對於現今的遊戲分類方式無法全然沿用過去在電影、書籍等傳統多媒體上的分類依據，在一些細節如風格上，它們之間的界定並不完全相同，因此有必要定義新的分類依據，亦即遊戲「類型」(Wolf, 2001; Apperley, 2006)。

遊戲類型是綜合考量主題、場景設定、畫面呈現或格式、玩家觀點與遊戲歷程策略等等後，再予以分門別類所決定的範疇。當前遊戲產業類別與書籍和電影的不同是，前者未必與故事、情節及場景設定相關，相反地，它專注於遊戲的遊玩方式，也就是風格(Novak, 2006)。通常我們會以遊戲裡的互動機制而非視覺、敘事差異作為分類依據(Apperley, 2006)。舉例來說，一款動作遊戲，無論外在的時空背景發生任何改變，我們會因為它的主要遊玩元素一致而一概視之為動作遊戲，這也是它與電影、書籍分類上最大的不同之處，也因此遊戲挑戰常用來當作定義遊戲類型的判斷因素。

然而遊戲類型的分類，無論在學界或業界並無嚴格且統一的規範和標準，Crawford(1984)在闡述自己定義的分類上也提及電玩遊戲設計是一門與時俱變的產業，當下所做的分類或許一下子將變成過時而不適用，但是確實分析遊戲類型可幫助我們釐清遊戲脈絡並明白設計方向。某個層面來說，也可以幫助我們了解市場走向，在業界和遊戲評論上常見有動作(Action)、冒險(Adventure)、格鬥(Fighting)、平台動作(Platformer)等各式分類(Moby Game, n.d.)。過去有學者針對市面、遊戲評論上的分類作出整理，Berens 和 Howard(2001)認為這些分類可以反映出業界對於不同遊戲的分類傾向與這些遊戲代表的行為模式，這些分類分別是：「Action」、「Adventure」、「Driving and Racing」、「First-Person Shooter」、「Platform」、「Puzzle」、「Role-playing」、「Strategy and Simulation」、「Sports and Beat-‘em ups」。

二、平台遊戲(Platform Game)

平台遊戲為動作遊戲的分支。動作遊戲(Action)屬於涵蓋範圍更為廣泛的較大集合分類類型，主要定義為需要高度反應力與操作技巧的遊戲，幾乎包含以上兩個要素的遊戲都能稱作動作遊戲(Berens & Howard, 2001)。而平台遊戲泛指這類遊戲群中其關卡提供一連串可供玩家佇足活動的平台，並要求玩家透過移動或通

過障礙物穿梭於平台與平台之間，以達成遊戲目標或進行遊戲活動。這裡的「平台」並非指各類硬體平台如電腦、手持裝置或作業系統如 Android 裝置或 IOS 平台，在本類型名稱上，此名詞是指可供玩家站立其上如臺階、地板這樣的環境地形，較為接近「平臺」泛指高出地面而寬平場所的名詞定義。這類型遊戲涉及大量在平台間移動跳躍的行為，包含在兩個懸浮平台間移動、越過障礙物等等，跳(jumping)是該類型遊戲最普遍且統一的操作玩法，偶而跳的動作會被其他如擺盪(swinging)或彈跳(bouncing)等動作取代，但這些都屬於跳躍的變化形態，依然屬於該遊戲類型下的範疇。它的一些傳統元素還包含跑步、爬牆或梯，它也時常加入及整合其他遊戲類型的元素衍生新的遊戲玩法，例如結合了射擊元素的魂斗罗(Konami, 1987)、帶有角色扮演色彩的「惡魔城：月下夜想曲」(Konami, 1997)、蘊含益智解謎要素的時空幻境(Blow, 2008)。

平台遊戲的歷史起源於 80 年代電子遊戲發展初期，從橫向捲軸(side-scrolling)遊戲一路開始蓬勃發展，這時期的經典代表包含「超級瑪利歐兄弟」(Nintendo, 1985)、「音速小子」(Sega, 1991)、「大金剛」(Rare, 1994)，而後隨著 3D 渲染技術成熟，90 年代中期開始一波波出現 3D 平台動作遊戲，第一款代表作品「瑪利歐 64」便於 1996 登上檯面，而後陸續不少 2D 捲軸遊戲被更新、改版成 3D 的平台遊戲，如「大金剛 64」(Rare, 1999)。

第二節 關卡生成(Level Generation)

本節將延伸關卡生成技術討論至內容與情境生成等多個層面，將從過去程序性自動生成技術的歷史與應用，談論至現今於遊戲上的功能用途，並輔以實際例子加以說明。

一、程序性生成(Procedural Generation)

程序性生成(Procedural Generation)屬於自動生成(Automatically Generated)技術下的分枝，是一種高度廣泛運用的技術，使用上具有高度的自由，其語法以及規則由設計者隨需求自行定義，可採取隨機或偽隨機(pseudo-random)亂數，常見用在建築、室內規劃(interior)、關卡設計與地形、植物、雲彩等自然景觀，也有應用於藝術創作等各方面領域。單就遊戲上，程序性內容生成便擁有多種設計用途，遊戲內容的生成除關卡生成(Level Generation)外也包含情境生成(Scenario Generation)等各個面上，涵蓋遊戲世界、場域、地形、人物、劇情、任務及各式遊戲配套內容。

程序性內容生成遊戲能為遊戲本身創造各種新的遊戲歷程，或許因為其特性與主張上一多樣的遊戲內容可引導玩家去享受到不同的遊戲經驗，進而增加玩家繼續遊玩甚至於破關後重玩的意願，程序性生成技術在遊戲中最常見的作用即是提高再玩性以增加一款遊戲的趣味和延長遊戲壽命(Smith, Gan, Othenin-Girard & Whitehead, 2011)。在 Elite (Braben & Bell, 1984)此遊戲中則是為了迴避當年的技術不足，無法把遊戲世界存放在硬碟及記憶體中。

程序性關卡生成技術可追溯到 80 年代的 Rogue，一款迷宮探索型 RPG，其中 Rogue 最顯著與創新的部分在於具有可無限推進的地域關卡(unique dungeon levels)，而後類似或具備 Rogue 主要遊戲特徵的遊戲都稱為 Rogue-like 遊戲，包括有地獄之門：倫敦毀滅(Flagship Studios, 2007)、暗黑破壞神 II (Bizzard North, 2000)、地底尋寶(Yu & Hull, 2013) 等，它們都使用了部分或改良了最初 Rogue 關

卡生成的技術。在這些遊戲例子中，最經典的莫過於矮人要塞(Adams & Adams, 2006)，它的地形、空間，甚至整個世界，一切都來自系統自動生成。

以遊戲設計師的角度而言，程序性生成使得遊戲設計有了更多靈活的管道。以往遊戲世界的物理參數、障礙物會在關卡設計前先做好設定，假使製作關卡中途需要更改障礙物的一些行為，可能得連整體大關卡都必須重新來過，但透過程序性生成演算，我們能在遊戲開發的任何時刻更動遊戲參數，讓系統自動因應每個關卡產生的新變化，甚至產生可以根據玩家行為作出調整的人工智慧。(Fisher, 2015)

程序性生成的隨機特性，為遊戲提供了多樣好玩的路線，更具體來說它能帶給玩家挑戰(challenge)、新的遊戲型態(new game types)、客製化(customizability)與一致性(uniformity)。以 Cloudberry Kingdom 這款遊戲為例，玩家與遊戲世界的互動會被記在系統參數裡，在下一回遊戲生成時系統會修正各個隨機因子的機率比再演算出適合玩家的難度，確保無論是新手或者厲害的玩家都能獲得挑戰上的成就，並且透過完全依賴隨機關卡生成的機制，將玩家記憶路線的破關策略導向實質的操作技巧。此外針對不同的玩家，系統可以根據玩家行為作出調整達到客製化，而對於遊戲本身，可以藉由微調參數產生一連串相似的關卡，使關卡的流動順暢、保持一致性，進一步維持玩家在遊戲過程的心流體驗。(Fisher, 2015)

近年來有越來越多遊戲內容的生成技術被提出，不論是學習與記憶玩家的喜好傾向甚至即時回饋、適應玩家在遊戲運行中的即時需求。在一些例子中，開發者導向(author-guided)的關卡生成會於遊戲執行前做好全盤管控，像是為了培訓遊戲做的情境場域(Hullett & Mateas, 2009)，偶而也允許遊戲運作期間進行部分編修，如文明帝國(Firaxis Games, 2005)。

自動生成技術的優點在於能免去大量的人力成本，甚至可能提供人工設計關卡所無法達成的無限提供玩家好玩的遊戲場域與遊戲內容，讓玩家花更多的時間在遊戲上面 (Compton & Mateas, 2006)。不過，Rogue-like 關卡生成技術雖然可以

透過人工設計的元件確保它的可玩性與彼此關聯的連結，但早期主要是針對具有一層層延伸的地域，類似地牢或迷宮的探索型關卡遊戲而設計的鬆散結構，這樣的結構是比較適合冒險 RPG 以及戰略遊戲。對於運用在平台動作遊戲上，我們必須額外注意到遊戲機制中保有的自然或非自然物理特性，進一步改良或制定新的架構，以避免程序內在的隨機因子會破壞原先遊戲預期的挑戰，例如意外產生玩家實質上難以跨越的巨大間格。

針對防範與降低遊戲設計中潛在問題的重要性，Fisher(2015)主張應該重視遊戲中關卡在整體遊戲設計上的意義與邏輯程序。關卡本身的存在意義來自於它能適時提供玩家必要的挑戰；一個具備挑戰性的關卡應該位於簡單易玩與極盡虐待(masochistic)之間的好球帶，如同心流體驗的規劃。

而配合嚴謹的設計規劃，需要制定良好的程序性演算；在設計上可以進一步檢視程序邏輯上是否符合並具備可行性(Feasibility. Can you beat it?)、設計有趣(Interesting Design. Do you want to beat it?)與合宜的技能等級(Appropriate Skill Level. Is it a good challenge?)等三項原則。也因此，Fisher 說明在製作 Cloudberry Kingdom(Pwnee Studios, 2013)時，基於這些考量為它設計了一套創建程序性生成關卡的演算法可以去適應每個玩家並改變遊戲參數。

過去，為了擴展自動關卡生成技術到其他遊戲類型上，Compton 與 Mateas (2006)考量到最初應用於遊戲的關卡生成系統，是針對具有延伸性地域的冒險與策略模擬遊戲，對於不同類型的遊戲有諸多限制，此外基於音樂節奏可幫助玩家進入心流狀態以及平台遊戲設計重度依賴節奏感的兩大論點，因此將其他領域的音樂節奏模式(rhythmic patterns)導入遊戲關卡設計，並提出新的專門針對平台遊戲的關卡生成的階層架構模型；包含組件(component)、模式(pattern)、區域元件(cell)、區域結構(cell structure)四個層次。

其中結構可再細分為主結構與子結構，如圖 2-1 所示。這個模型說明了一款遊戲會有主要的關卡架構，像是圖 2-2 中常見的關卡設定，而在主架構下會有次

要結構，例如遊戲中除了主線劇情外還常會包含支線任務，模式則是用來組織遊戲關卡的內部組件；無論整體遊戲結構是屬於線性或非線性，模式僅闡述各個關卡段落中較嚴謹的線性玩法，以地底尋寶為例，遊戲中每關都有起終點的設定，玩家無論是如何破關，這段路線上基本是線性的。

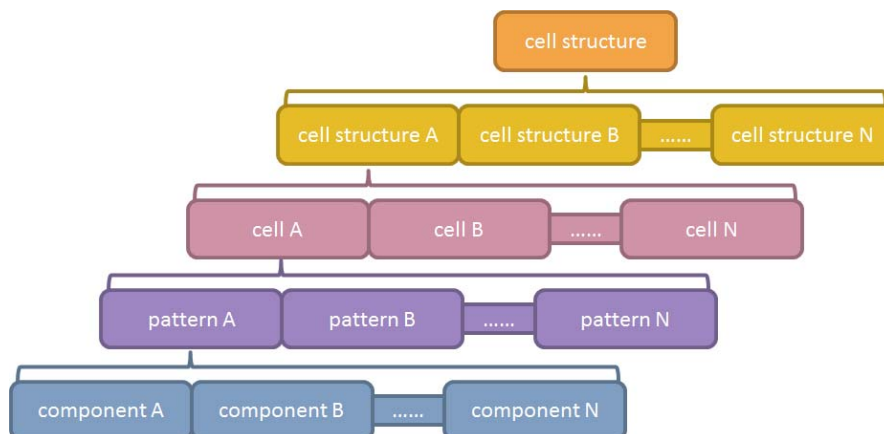


圖 2-1 Compton 與 Mateas 的遊戲關卡階層架構模型

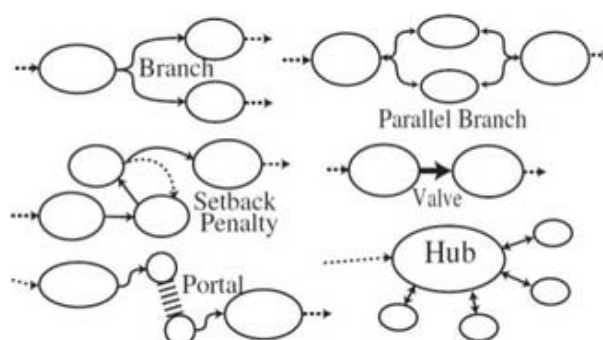


圖 2-2 遊戲設計的區域結構

在這之後發展的技術，各個利用系統自動生成關卡的遊戲也紛紛提出各自針對遊戲面設計或改良過往的模型。例如 Jack Benoit(Benoit-Koch, 2014)製作者將關卡生成重心放在相對於 Compton 與 Mateas 的模型底層之組件上，把生成架構以圖磚(tile)為單位分成背景、平台、梯子、裝飾小元件四個層次。而 Martin, Schatz, Hughes 和 Nicholson (2010)於他們設計的軍事訓練嚴肅遊戲中，將串聯遊戲整體脈絡的情境腳本模組化，並定義做為生成模型中核心的場景腳本(scenario)必須包含訓練目標(training objective)、基準線(baseline)、擴增內容(augmentation)和情境

故事(vignette)四個項目，如圖 2-3 所示；圖中淺藍色標示為生成對象主體也就是場景腳本，深藍色為腳本中主要必備的四個項目，其他白色則是次要與附帶項目。

針對 2D 平台遊戲上的生成技術研究，Smith, Treanor, Whitehead 和 Mateas (2009) 更引用 Compton 與 Mateas 提出的模型，以加入節奏群(rhythm)的概念設計了一套生成關卡的系統，並進一步於隔年完成關卡編輯器；提供設計師自訂遊戲節奏，使系統依據跑跳的節奏產生前方的道路，根據不同變化可能平順、可能有懸崖或者遭遇敵人挑戰等各種情況。

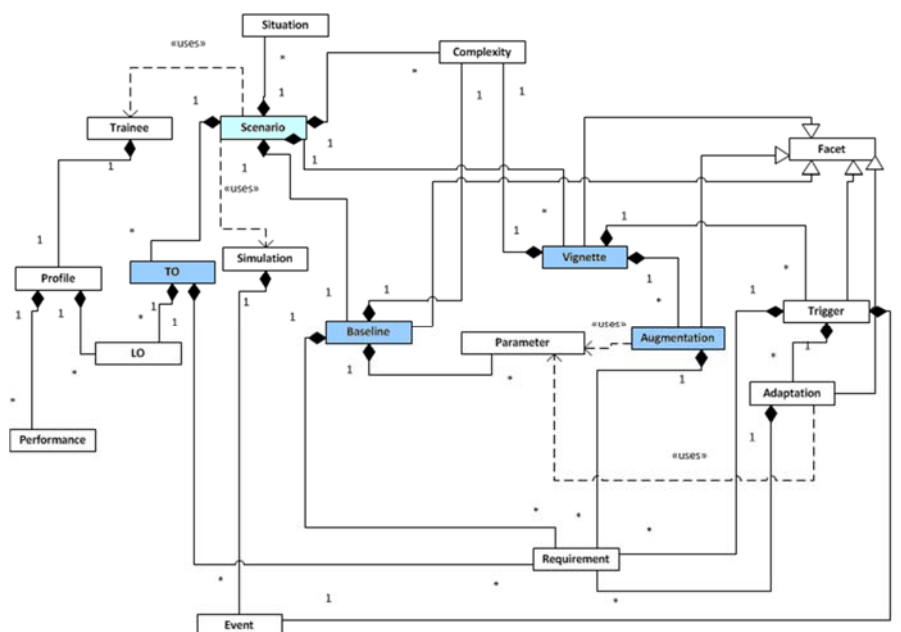


圖 2-3 Martin 的腳本模型架構

二、程序性內容生成遊戲(PCG-games)

市面上有眾多涉及程序性內容生成的遊戲，除前一節提及的暗黑破壞神系列，以及其他古老開源的 NetHack(The NetHack Dev Team, 1987)、Angband(Angband Development Team, 1990)等 Rogue-like 遊戲外，包含第一人稱射擊遊戲邊緣禁地 2(Gearbox, 2012)、極地戰壕 2(Udisoft, 2008)、惡靈勢力(Valve Corporation, 2008)、僅有 96kb 的.kkrieger(Farbrausch, 2004)，以及策略模擬遊戲

Spore(EA-Maxis, 2008)，和深受太空模擬遊戲 Elite 影響啟發的多人線上遊戲 Infinity: The Quest for Earth(I-Novae Studios, 2010)與獨立遊戲 Notis(Ghignola, 2000)，還有早期系列遊戲上古卷軸，尤其其中的上古卷軸 II 匕首雨(Bethesda, 1996)最具代表，另外其他類型遊戲還有即時戰略與回合制策略遊戲，例如 Tribal Trouble(Oddlabs, 2005)、X-COM 系列(Firaxis Games, 2012)、Planetary Annihilation(Uber Entertainment, 2014)等等，可以看到在各類遊戲上運用的範圍非常廣泛。

本節將以關卡中空間規劃的視角，挑選部分公開程序性生成原理的遊戲或以此借鏡衍生的關卡設計辦法，介紹並闡述這些關卡的生成方式。

1. 地底尋寶

這是一款號稱永不重複關卡的橫向卷軸遊戲。在網路上可找到 Kazemi 撰寫的網頁版，其中說明了部分關卡生成機制如何在此款遊戲運作，並釋出一小塊關於遊戲中解決路徑問題並產生關卡地圖的生成原理，以下將圖文搭配進行說明。

地底尋寶的地圖生成程序分為兩個階層架構，一個是大局面的關卡外貌，也就是棋盤(grid)內容，另一個是房間內部磚牆(tile)設定，亦指用於定義房間中每一塊磚牆是實質的牆壁、寶物、陷阱或空地等的模板(template)。

一開始定義此關卡是由 16 個房間構成 4×4 的棋盤如圖 2-4，該圖描繪了一個關卡基本的地圖全貌，可看到每間房間的左上角標有 0 到 3 不同數字，這些數字便是此房間的樣式編號。這些房間共有 4 種基本樣式(type)可供選擇；0 代表角落房間，用於填補路徑解答(solution path)以外的區域，1 指這個房間有左邊及右邊的出口通道，2 表示這個房間有左邊、右邊及下方的出口，而如果 2 的上方連著 2，那麼下方的 2 會額外打通一條往上的出口，最後 3 則是左邊、右邊與上面都有出口的倒 T 字房間。

製作一個完整的關卡地圖前，首先必須決定一條解決路經，也就是像一般設計迷宮時都會存在一條可行的路線。遊戲關卡地圖設計上，同理必須保證玩家從

起點到終點絕對有一條絕對可以走的路，但基於可玩性與提升重玩價值，同時也不希望這條路經是固定不變，常採取一些半隨機性的生成手段。在地底尋寶中，它的生成流程一開頭選定起點必然屬於最上排的某一間房間，而這間房間的樣式編號一定是 1 或 2 其中一個，如圖 2-5；可以看到作為起點的門隨機產生在這個房間某處，以紅色圓圈處所標示。



圖 2-4 地底尋寶的地圖生成範例

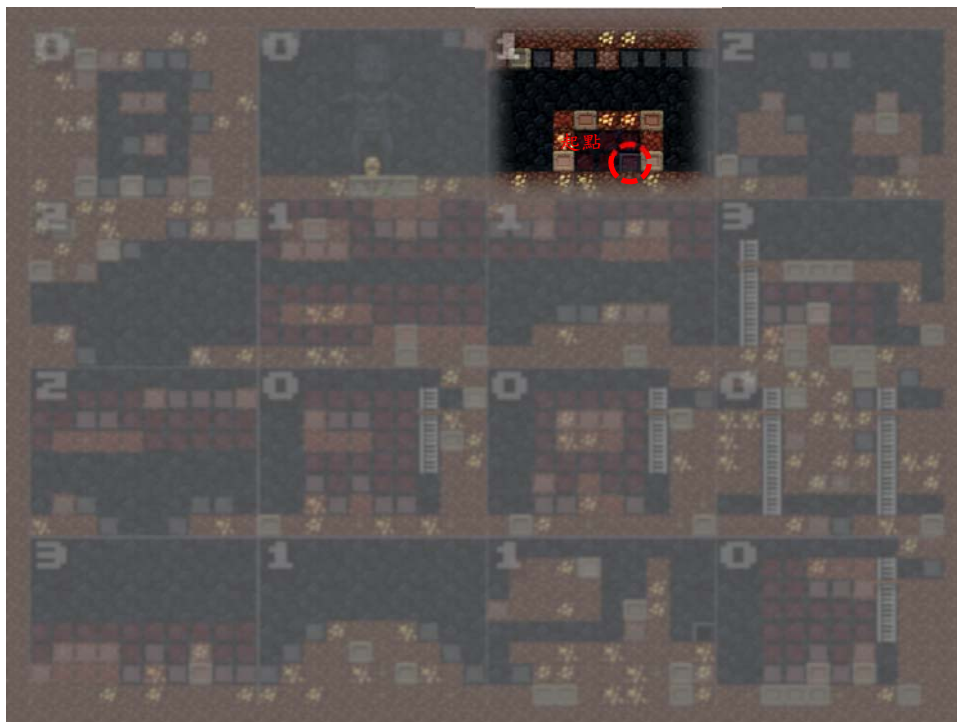


圖 2-5 地底尋寶的起始房間生成範例

決定起始房間後，便從房間的左、右或下方隨機選擇一個路徑的方向，機率比約 2：2：1，如果選擇左或右邊時發現會撞到關卡邊界，則無條件選擇往下。房間每一次只會生成一間，以此一間間推移路線前進。每次生成房間的初始樣式都定為 1，當決定下次的生成方向為往下時，則將樣式變更成 2，而當上一次生成的房間樣式為 2 時，該次的樣式改為 2 或 3，最後當生成房間來到最下排又試圖往下產生房間時，我們將這個房間定為終點房間，並把出口產生、結束路徑生長。

整個生成結果可以參考圖 2-6，圖中以紅線標示出起終點的生成路線，一條保證玩家絕對能從起點抵達終點的路徑，右下的紅圈則表示終點出口，玩家可從這裡進入到下一關；它的起點一樣，會隨機產生在指定房間的某處。

一些程序性內容生成的遊戲因為無法確定程序產生的地圖是否能玩，例如在撰寫關卡生成程序時並沒有把存在路徑解答之類的重要問題考慮進去，因此必須在地圖生成後反覆驗證，導致良率低落而無法即時生成。



圖 2-6 地底尋寶的路徑生成範例

關於路徑產生的方式，在 Jack Benoit 遊戲中有另一套類似做法，該方法中也同樣先決定出地圖棋盤，然後隨機決定起始與結束的房間，便開始生成路徑，只是它選擇捨棄空白的部分，僅留下路徑存在的房間。而對於每間房間之間的路徑，它選擇以隨機的方式決定每間房間必經路線上的點，然後按順序連上點與點之間的固定路徑，最後開始生成房間時，按照路徑在路線上產生台階、梯子，接著再由上而下、由左而右填滿房間內部其他的黑色區塊。整個過程可以參考圖 2-7，從左而右表示其關卡產生過程。

但此遊戲生成方式有個很大的缺點在於最後填補房間內部區域的演算上容易產生無用的台階且無法避免；這是因為固定路徑上的物件生成與其餘部分的生成演算幾乎是獨立分開，彼此互干涉下的程序演算設計並沒有細緻考慮如何銜接周遭環境與主路徑。

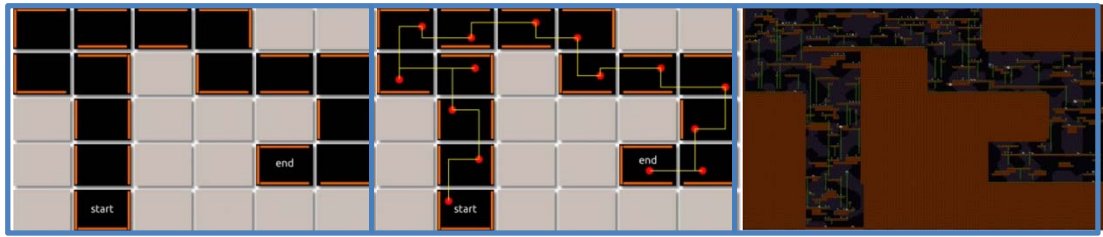


圖 2-7 Jack Benoit 遊戲關卡生成範例

回到地底尋寶關卡生成部分，在完成路徑解答之後我們用編號樣式 0 的房間把剩下的空白區域填滿，並且在樣式 0 的房間偶然成 3 或 4 個垂直排列時，給予個機率會更替成如圖 2-9 中 7、8、9 或 7、8、8、9 的蛇窟，然後預設這個洞穴底層充滿許多會殺死玩家角色的蛇與一兩個重要的寶物，讓這個關卡有更豐富的可能性，以及加深玩家往各個區域探索的動機和挑戰成就。

不過，這個方法其實同 Jack Benoit 一樣容易產生一個問題，例如下圖 2-8 最左上角的房間因為四周都被填滿咖啡色的磚牆，形成孤島一般的封閉區域。在一般的平台動作遊戲中，這樣的區域因為玩家難以意識到或就算意識到也會因過不去而成為多餘而沒有意義的區塊，所幸地底尋寶這款遊戲本身有個機制能讓玩家炸開地形開路，讓這個問題反而變成遊戲特色，如隱藏的秘密房間。

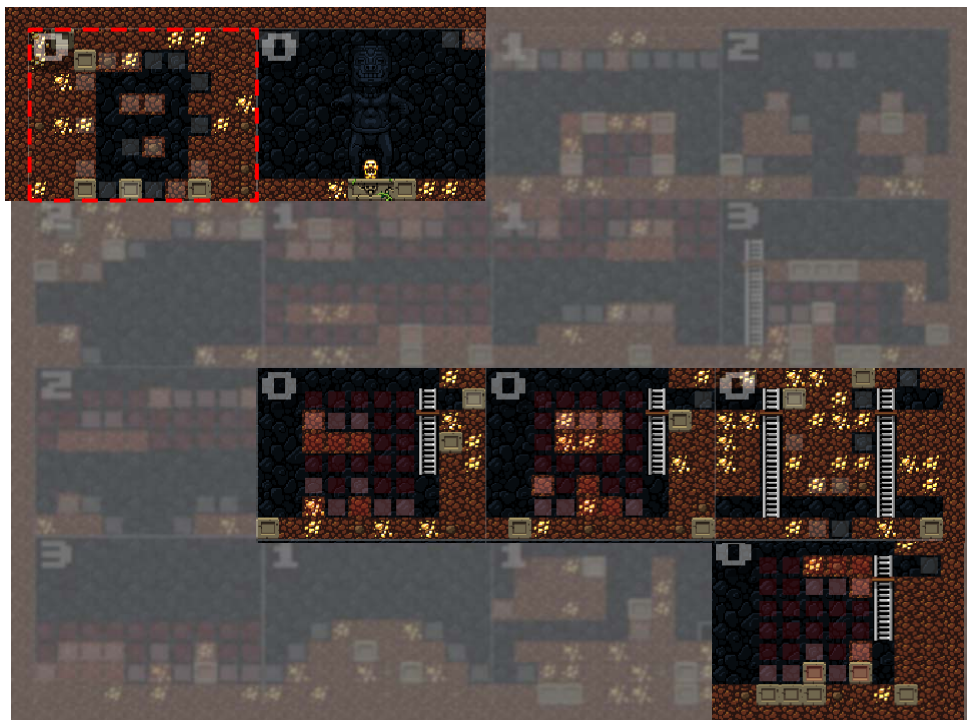


圖 2-8 地底尋寶的地圖填補範例

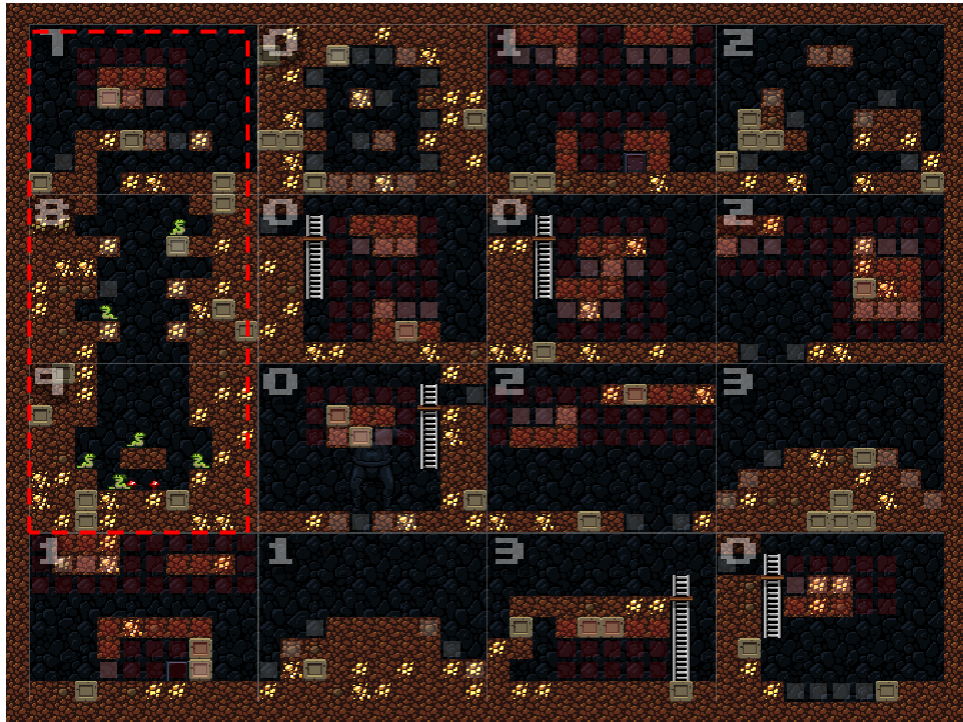


圖 2-9 地底尋寶的蛇窟生成範例

接下來在房間模板的部分，每個房間依據不同房間樣式，有數組不同的模板(template)可供隨機選擇，這些模板會定義成如圖 2-10 左側一樣的 10×8 陣列，從範例陣列中可看到標示各種磚與機率狀況的字元，這些字元的數量與種類也可能依據實際需求增減。在程序生成時，系統會根據讀取的字元決定應該產生的磚。

每一個模板會由一群靜態磚(static tile)、機率磚(probabilistic tile)與障礙磚(obstacle blocks)組合而成，靜態磚中 0 代表空地(empty space)、1 表示實心方塊(solid block)、L 指爬梯(ladder)而 P 為爬梯最頂端的台階，而機率磚裡包含各種機率狀況的相對應字元，例如 4 表示有 25%機率在爬梯旁出現推箱，其他各種可能狀況有 33%機率出現死亡尖刺等，障礙磚的部分則有代表躺在地上的 5 以及浮空的 6，一個障礙磚固定為 5×3 的大型磚塊，通常會由靜態磚和機率磚兩者相互拼接組合，如圖 2-10 右的陣列。

這些模板可說是遊戲設計發揮的地方，一個模板好壞會直接影像遊戲好玩與否，例如兩邊懸崖間是否有相連的台階或梯子的下方有沒有陷阱等，這些設計都關係到遊戲整體難易度與流暢性，並直接反應在玩家實際的操作上，假使遊戲需要的操作技巧比預期的高出許多，玩家可能會覺得這個關卡設計太難、不好玩，因此必須好好規劃每個房間能擁有的元件數量與種類，再透過模板安排達到半隨機的關卡設計。

在圖 2-11 中可以看到以同樣樣式編碼房間構成的關卡地圖，可能展現不同的地形外貌。圖左跟圖右分別擷取兩種可能狀況做為比較，可看到如綠框標示處，基於不同的模板，同樣式編碼的房間會有不同的樣貌，而同樣式與模板的房間如藍框所示，因為受機率磚的影響，也未必一模一樣，其他如紅框可以看到障礙磚不同的變化。

1100000000	00000
40L6000000	00102
11P0000000	71177
11L0000000	
11L5000000	
1100000000	
1100000000	
1111111111	

圖 2-10 template 與 obstacle blocks 範例



圖 2-11 地底尋寶的同樣式地圖生成比較範例

2. 憤怒鳥(Rovio Entertainment, 2009)

這是一款具有物理特性的益智遊戲，玩家透過彈弓發射小鳥來打破層層建築物與豬。關於遊戲內部一關又一關的建築物與豬如何布局，網路上可以找到針對此遊戲分析後簡化的關卡生成公開模型，以下藉助此模型進行說明。

該模型分為兩個部分，第一個部分為依據參數表生成關卡，第二部分為更新參數表，如此交替生成布局與更新學習循環數次後，將產生遊戲平衡良好的關卡。

首先從生成關卡布局的部分開始進行說明，它總共分為三個步驟；依序為決定一個關卡中有多少個欄(column)，再來每個欄中間的間距為多少，最後選擇每一欄中間的物件數量為多少。這裡的欄指關卡中的水平方向上會有多少層障礙物或空地，以圖 2-13 來說它總共有五個欄位，第一和最後一欄裡沒有裝任何物件，所以會呈現出空地，中間三個欄則各自以欄內的物件堆疊成一個個建築塔，這些塔彼此間該相距多少，由每欄中間的間距數值控制，而每一欄會裝入多少物件我們讓參數表控制，並預定一個會裝入物件的欄位裡必然會有一個頂部(top)和一個底部(ground)物件，剩下的中間(middle)物件數量便直接影響這一欄的物件堆疊高度。針對這些各式各樣的物件，我們定義了一個物件表單如圖 2-12，裡面從 1 到 22 對應豬與不同大小的圓形、矩形與三角形的木材或石材物件的編碼。

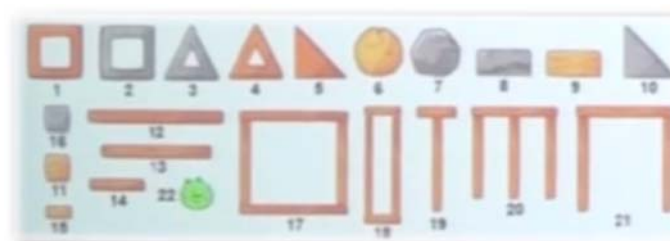


圖 2-12 憤怒鳥的關卡生成物件表

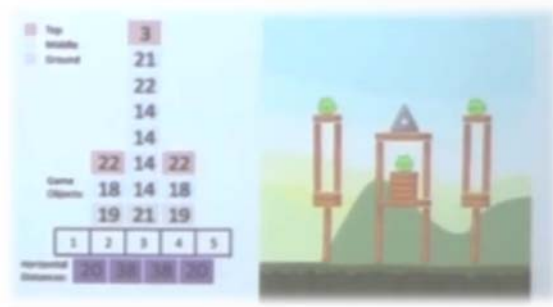


圖 2-13 憤怒鳥的關卡生成範例圖

接著，在機率表的部分，我們透過一系列預計算的表格決定每關中每個物件的出現機率，以及每個欄位中會疊幾層物件與這些堆疊好的建築塔彼此的間距又有多遠，這些參數表分別可參考表 2-1 到 2-3。

在表 2-1 中標示了每種物件在上、中、下層各自出現的機率，可以看到代表豬的 22 以及三角形 4、5、10 與圓形 6、7 這些無法當基底的物件普遍在中、底層的機率為 0，這是因為遊戲設計上必須確保場景中的建築塔一開始是處於穩固的狀態，不然玩家開始動作前建築便可能自行倒塌，而在表 2-2 可以看到各個欄位可能被堆疊的物件數量機率，這個表會直接影響建築的外型高度，以欄位 4 而言，100%不會被放入任何物件，將呈現一塊空地，最後表 2-3 會影響整體建築物群的分布鬆散或緊密，表中標示每個欄位紀錄彼此可能的間距機率。

一開始這個表會由設計師自訂而後由系統更新，在生成關卡時系統便透過讀取這個表格計算出這次生成的結果，結束生成後系統會透過公式(1)評估這次生成出來的所有關卡，然後從中挑平衡性最好的一個當學習對象，以此更新參數表再重新生成關卡。每次更新學習時僅從對象身上學習一點點並在新生成後重新挑選學習對象，透過逐次學習後我們會得到各個參數機率比趨向平衡的優化參數表，到此系統便能確實生成一個在驗算上具有平衡性的遊戲關卡。

表 2-1 憤怒鳥關卡生成的各種類物件生成機率參數表

Game Obj.	Ground	Middle	Top
1	0.003	0.3	0
2	0	0	0.02
3	0.17	0.1	0
4	0	0	0
5	0	0	0.18
6	0	0	0
7	0	0	0.1
8	0.05	0.12	0
9	0.05	0.18	0
10	0	0	0.05
11	0	0	0.15
12	0.1	0.2	0
13	0.1	0	0
14	0.1	0.2	0
15	0	0	0.1
16	0	0.1	0.1
17	0.1	0	0
18	0.1	0	0
19	0.1	0	0
20	0.08	0	0
21	0.02	0	0
22	0	0	0.3

表 2-2 憤怒鳥關卡生成的各欄位物件數量機率參數表

Objs. stacked	Col.1	Col.2	Col.3	Col.4	Col.5
2	0.4	0.2	0	0	0.7
1	0.4	0.6	0.1	0	0.2
0	0.2	0.2	0.9	1.0	0.1

表 2-3 憤怒鳥關卡生成的間距機率參數表

Dist.	1-2	2-3	3-4	4-5
[60,80]	0.25	0.0	0.1	0.2
[40,60]	0.25	0.0	0.0	0.4
[20,40]	0.25	0.5	0.1	0.2
[0,20]	0.25	0.5	0.8	0.2

以下為憤怒鳥關卡生成中用於評估關卡的簡化 fitness 公式：

$$f_{ind} = \frac{1}{n} \sum_{i=0}^{n-1} v_i + \frac{\sqrt{(|b|-B)^2}}{Max_b-B} + \frac{1}{1+|p|} \quad (1)$$

第一個項目考慮關卡內物件的平均速度 v ，用於量化關卡穩定度，如果物件初速度大於 0 代表關卡生成時擺放的物件不穩固，可能有遊戲開始前便傾斜倒塌的問題，第二個項目為向量化預期物件 B 與所有物件 $|b|$ 的距離，這個數值可以反映出這個關卡中布局的建築物群鬆散程度，最後一個項目是針對關卡中豬的數量 $|p|$ 進行評估，如果豬越多、算式得出的數值越小。整體而言，一個好的關卡在此公式評估上數值會趨向 0，參數表便是取數值最小的對象做為學習更新的依據。

3. 超級瑪利歐兄弟

這是款經典的橫向卷軸平台動作遊戲，其遊戲關卡順暢而有趣，在 2009 的 Wii Sports 誕生前它維持了 20 多年最暢銷的電子遊戲紀錄。因為這個緣故，在此之後開發的橫向卷軸遊戲上，往往希望能從瑪利歐這款遊戲中延續下好玩的關卡設計，也因此需要一套規則將感官上的設計美感轉換成程序語言。在 GDC 2015 的程序化關卡生成之案例分享中，戴文凱教授分享並說明了一種以現有遊戲作為參考對象、運用馬可夫鏈作為定義生成規則的手段，在下圖 2-14 中可以到以

$P(S_{t,r} | S_{t-1,r}, S_{t,r-1}, S_{t-1,r-1})$ 作為範例的圖示化說明。

馬可夫鏈很適合用在邏輯規劃，可以幫助我們從現成的遊戲中提取出一套可依循的設計規則，例如我們想擷取瑪利歐遊戲關卡中某一段有趣的設計，並模仿出接近的遊戲歷程，便可以借助馬可夫鏈分析玩家操作時所做的一連串跑跳動作與遊戲關卡變化之間的關係、整理出相對的關聯邏輯。像是在某個障礙地形中，玩家角色在跳之前必須跑，跑之前可能會有先跑或跳等動作行為，而這些行為會分別對應到遊戲畫面上各種情況，根據這些資訊便可從中歸納出一套邏輯規則，以仿照出一個相似的遊戲歷程，達到程序化產生關卡。

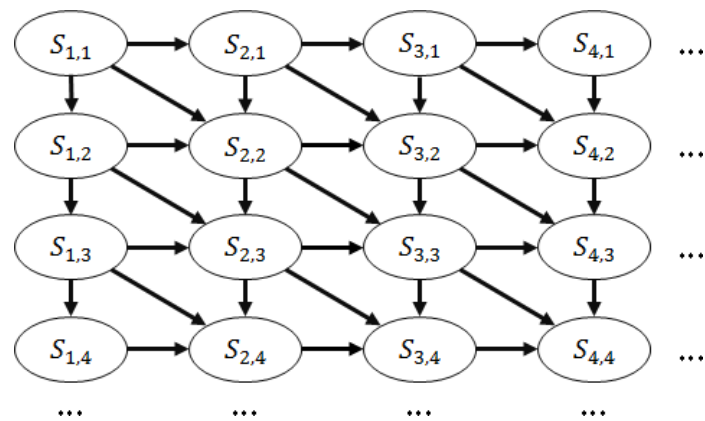


圖 2-14 馬可夫鏈(markov chain)

在實作上我們可以利用資料庫探勘的做法來分析遊戲關卡並在關卡生成時做資料檢索。為了增加效率，尋找最短路徑、簡化與加速分析速度，現今已有許多用於搜尋與解決路徑問題的演算法被相繼提出，我們可以使用如圖 2-15 中表示的演算法，來分析玩家角色移動情報，並進一步連接跟其餘關卡內容物之間的互動關係，例如：道路通不通，玩家撞到敵人會死掉等等。根據規則我們把現實中的遊戲圖像與玩家情報轉化為位元表，如圖 2-16 代表世界情報的二維地圖(map representation)，而後我們便能根據這個地圖索引產生新的關卡。

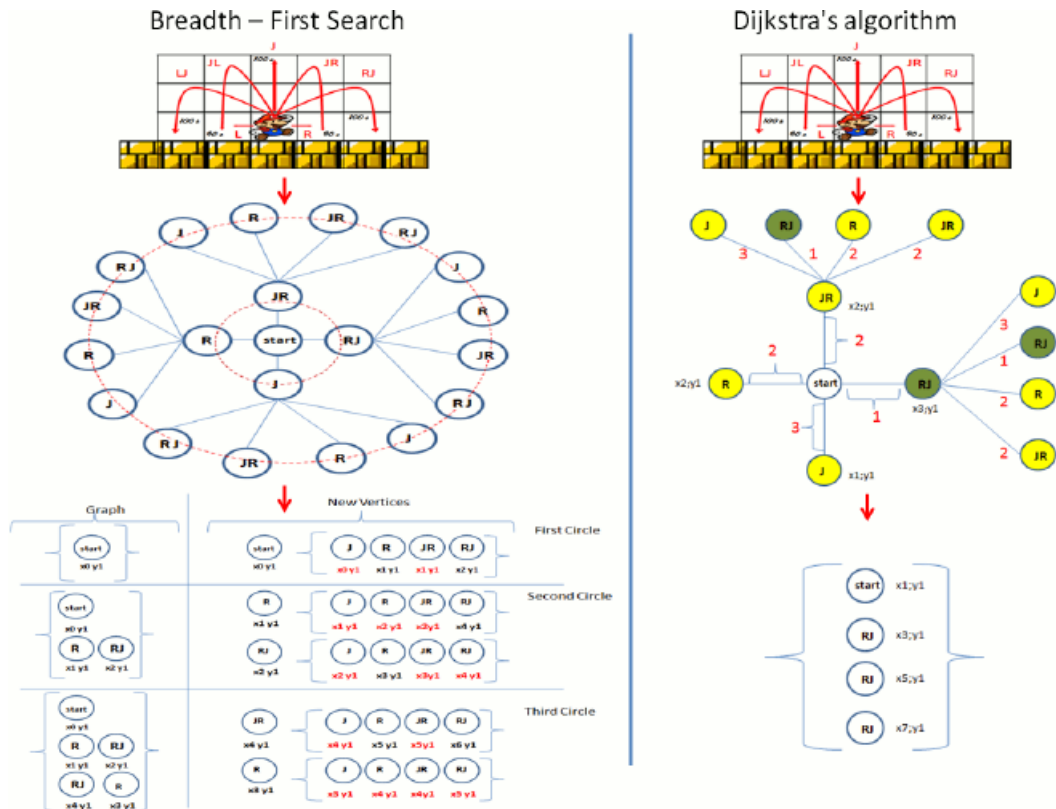


圖 2-15 Breadth-First Search 與 Dijkstra's algorithm



圖 2-16 範例 The Mario Map Representation (Adelhardt & Kargov, 2012)

在關卡生成中我們可能會碰到些預期外的狀況，例如我們參考的關卡對象碰巧沒有特定的遊戲事件，而我們又希望它能夠出現或者在程序邏輯編寫上沒有預防它可能發生，因此無法用既定規則預分析下一步該如何前進，或者有可能是之前分析得到資訊量不足、沒有考慮到一些彈性狀況，如果演算上能夠把條件適時放鬆或許就能解決。

基於這些緣故，也為了避免未知的狀況破壞遊戲，在這套以瑪利歐為範本分析設計出的生成程序中，設計者於演算處理上搭配使用 Look-ahead strategy 與

Fallback strategy 兩個方法，讓產生無法預期的錯誤時，能回到前一個步驟重新來過或者改用較通融的規則，如果這樣問題依然無法解決則用半隨機的方式產生最後結果，以此避免未知的錯誤情況。

另外考量到實際遊戲上，關卡走向固定為從左往右的橫向設計，且遊戲世界具有向下重力以及玩家的基本認知上，遊戲世界跟現實世界會有類似的物理特性，因此預設遊戲內各個物件及背景的磚塊生成順序為從下到上、從左到右，以避免部分物件如煙囪不自然的長在半空，並且默許道路的規劃上可能有玩家無法回頭的狀況，但不會因擺放相反導致一開始就過不去。

在圖 2-17 中可以看到為 GDC 2015(引用)的程序化關卡生成之案例分享的關卡生成結果，圖中以程序性生成的關卡已具有與瑪利歐遊戲相似的設計成果，但也可以看到還有些待解決的問題如紅圈處，如煙囪過高與不自然的堆疊。



圖 2-17 程序化關卡生成案例

第三節 關卡與遊戲設計(Design Pattern of Game Level)

本節將針對關卡與遊戲設計做回顧探討，說明關卡設計須注意的地方，同時由分析過去關卡設計的架構，決定本論文依循參考的模型對象。

一、關卡設計(Level Design)

關卡提供玩家互動與探索遊戲世界規則的場所和方式(Byrne, 2005)。作為遊戲的容器，關卡設計會直接影響遊戲本質好玩與否，是遊戲設計過程中至關重要的一環(Byrne, 2005)。延伸 Sid Meier 的話，遊戲設計的目的是尋找與創造“一連串有趣的選擇(interesting choices)”。在這個基礎上，關卡設計本質上屬於一種工藝(craft)，我們需要些輔助工具，確保設計中的關卡能銜接上最終玩家在遊戲世界中的遊玩體驗，同時降低開發的過程，縮小測試階段上可能面臨的問題(Larsen, 2006)。不過，這些工具基本上是作為設計輔助，設計師不能完全依賴工具，像是一些美感、人性化及有趣的感官設計，就如 Nelson 與 Mateas(2007)所說提倡的，關卡設計的目標不完全是為了取代設計師，相反的應該進一步了解更深層的設計問題，以提供能支援設計師的智能設計工具。

關卡設計其實十分複雜，隨著不同遊戲類型、機制內容，關卡結構上的需求與層次差異甚遠。過去不少研究著墨在尋找通用的關卡設計準則，有許多設計法則被相繼提出，但關卡設計與個別的遊戲類型有很深的關聯；遊戲設計的通用書籍通常不會直接論及關卡設計這一區塊，例如 Salen 和 Zimmerman(2004)在討論以經驗規劃遊戲(crafting the play of experience)時，便悄悄帶過重複與互動兩個關卡設計中的重要概念，這也是因為許多的規則與挑戰是從類型分析下提取發現的(Smith, Cha & Whitehead, 2008)。

為了達到關卡設計的目標且考量到遊戲種類繁多的狀況下，以單一類型下去分析現有的關卡設計元件是比較可行的方式；如 Nelson(2007)對打磚塊風格的遊戲(Breakout-style Games)透過一連串分析提出一個還原視圖；藉由歸納重組，將數款該類型遊戲分解還原出該類型遊戲所需具備的元素以及組織架構，Smith 等

人(2008)則延伸此分析手法用於深入剖析 2D 平台遊戲並提出通用的項目，這些學者們透過分解關卡提取出該遊戲類型的主要成分，並訂定規則好重組這些元件成為有趣的設計，提供設計師對這類型遊戲關卡設計上的方向。

一些其他的研究如 Boutros(2006)以暢銷流行遊戲分析視覺(visuals)、控制(controls)、結構化挑戰(structuring challenges)對遊戲的影響，幫助設計師製作受歡迎的遊戲。而關於將某類遊戲拆解成其運行要件，並藉此分析重組的做法，在其他研究例如 Bjork 和 Holopainen(2004)的遊戲活動框架(activity-based framework)也可以看到類似的架構，但從這之中可以發現遊戲設計跟關卡設計在根本上有些微不同，雖然它們存在許多相似或重疊的性質，也都可依循破壞分類重組的分析手段，但關卡設計會更著重於結構與時間方面(structural and temporal aspects)的設計面向。

另外，從 Zagal, Mateas, Fernandez-Vara, Hochhalter 和 Lichti (2005)關於遊戲工程論述(Game Ontology Project)明確的看出關卡設計與遊戲設計彼此分屬不同層次這一點，他們將遊戲切割成由“關卡”(level)、“波”(wave)、“檢查點”(checkpoint)構成，根據遊戲挑戰有時波也會以拼圖(puzzles)替代，很明顯地此種分割把空間布局與遊戲挑戰區分為關卡與波或者拼圖之下，不同於關卡設計研究傾向把空間布局與遊戲挑戰兩者整合起來一併討論。Byne(2005)主張挑戰是關卡設計中最重要點，因為它通常是左右玩家享受遊戲的關鍵。基於遊戲類型一致與研究目標較偏向關卡設計之下的程序生成研究，本研究在設計關卡模型的選擇上將參考並沿用 Smith(2008)等學者提出的分析模型。

二、平台遊戲關卡設計框架(Framework)

在 Smith 針對 2D 平台遊戲提出的模型中，可分成組成元件與結構示意圖兩個部分；這些元件因為其性質與意義有其重要性，或者說為了體現此類型遊戲的特色，這些元素是必要的存在，因此個別獨立出來作為組成遊戲關卡的重要元

件，而示意圖的部分則用來圖像化說明如何組合元件以及整體架構關係，用於組織化生成或設計關卡。相對於較注重關卡視覺設計的研究，此模型則著眼在底層關卡結構，也因此分類上僅考慮元件功能面的目的導向，而省略在視覺象徵上的表現意義。整體模型架構如圖 2-22 所示，在元件的部分則分為以下五個項目：

1. 平台(platforms)：泛指玩家在關卡遊戲過程中跑、跳其上的物件對象，其形態有單調的平面(surfaces)、循環迴圈(loops)或某些物體如方框的頂部(tops of item boxes)，具體的例子如下圖依上述順序由左至右排列的代表性平台地形。一些更細節、特殊的平台設計還包含隱形的(invisible)、無法久站臨時的(temporary)或某些遊戲獨有自創的，例如只能按規則踩踏不同顏色的台階，踩錯會觸發陷阱或傳回原點等等。



圖 2-18 瑪利歐 3 關卡內的平台實例

2. 障礙(obstacles)：遊戲中能給予玩家角色(avatar)損害或危機的物件統稱，常見的有敵人角色與死亡陷阱等，包含靜態與動態的各式挑戰，在這層定義中平台與平台間的間隙也被認為是障礙的一種，雖然這在關卡中並不是那麼明確存在的實體物件。



圖 2-19 瑪利歐 3 關卡內的障礙實例

3. 移動輔助器(movement aids)：指任何存在關卡中幫助玩家移動的物件，通常使用角色跳躍或跑步以外的特殊方式運作，如梯子(ladder)、彈簧(springs)、彈簧床(movable trampolines)以及繩索(ropes)等。一些特殊的例子像是瑪利歐裡可以使玩家跳得更高的彈力磚塊(jump block)便具備平台與移動輔助器兩個特性；這也說明這些元件未必會獨立存在，甚至可以多元組合成多功能的元件。下方圖中紅色標記了瑪利歐遊戲裡用來跳高的 jump block 與攀爬用的藤蔓

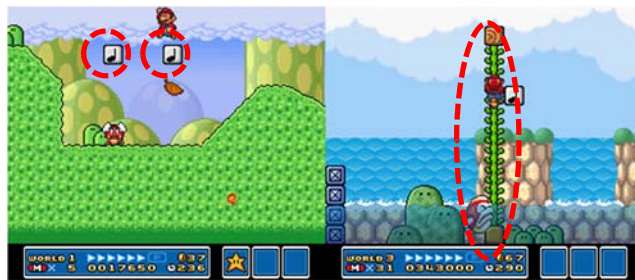


圖 2-20 瑪利歐 3 關卡內的移動輔助器實例

4. 收集品(collectible items)：此物件為提供玩家遊戲中的獎勵，它可能被設計成金幣、加持能力的能量球或增加生命的道具與積分回饋等等，因為收集品具有驅使玩家行動的誘因，也常被用作引導玩家，例如下圖圖左中將金幣置於懸崖的一側，以隱性的提示透漏遊戲路徑。



圖 2-21 瑪利歐 3 關卡內的收集品實例

5. 觸發器(trigger)：表示任何用於顯示關卡中各種改變狀態的特殊物件。舉例來說，像是一些解謎遊戲常看到場景中有用來切換平台位置或觸發事件的特殊機關，或如下圖圖中左右標示觸發前後結果的瑪莉歐遊戲中把磚塊換

成金幣的轉換開關(P Switches /Switch Blocks)，諸如此類可能改變環境甚至改變玩家角色的引發物件便稱為觸發器。



圖 2-22 瑪利歐 3 關卡內的觸發器實例

在結構面上，Smith 的模型說明整體遊戲關卡為由構成關卡挑戰的元件組織出的階層結構。一個關卡分為主要核心(cell)與連接的橋樑(portal)，在這之下是節奏群(Rhythm Group)與其包含的元件要素，關於這些物件與構成要素的關係，可參考圖 2-23 圖式化的遊戲結構設計模組。

其中，針對節奏群在關卡設計中扮演的角色，Smith 在研究中有提及並沿用 Compton 與 Mateas 當初引進音樂節奏研究的思維至遊戲設計上，說明由這些元件可組合一小段挑戰，例如跑上去跳過懸崖，而為了使這些挑戰有意義，玩家玩起來順手，將這些一小段一小段的挑戰段落做節奏化拼湊，形成一個較長的挑戰節奏，而這些不同挑戰組合出的節奏可再組合出更大的節奏群，就像音樂可由不同的幾個旋律反覆交織組成一首曲子；我們可以把節奏群視為一個節奏化的遊戲挑戰集合，它表現出遊戲中某一段落關卡中玩家會面臨及經歷的挑戰。

屬於上位階層的關卡核心與橋樑彼此相互連接以構築遊戲的線性玩法(linear gameplay)，而回過頭看每個關卡核心則由一到複數個節奏群提供或多或少、非交集(non-overlapping)的關卡元件，組成一系列的遊戲挑戰；可以說下位階層是用來提供遊戲挑戰，而上為階層是用來組織遊戲路徑。

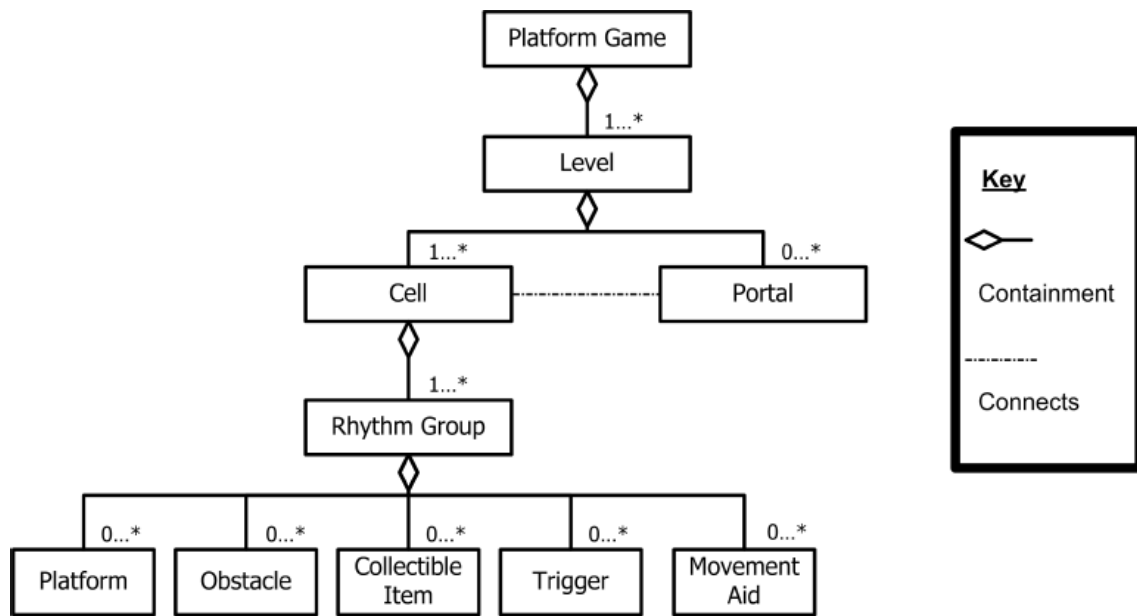


圖 2-23 Smith 的關卡架構概念模型

第四節 L-system

L-system 是 Lindenmayer 系統的簡稱。L-system 是一系列使用不同形式正規語法規則來定義分支構造如何改變的系統(Prusinkiewicz & Lindenmayer, 1990)。定義上，它具備自我遞迴的生長規則，於一開始給定系統初始值而後根據定義之生長規則去做相對應的生長；可以把它看作一個會隨時間成長推進的系統。其基本概念來自字串改寫(string rewriting)，根據規則或指定流程變換輸入內容，並且藉由回傳迴圈(feedback loop)產生遞歸結果。

L-system 將重寫此一概念發展成嚴謹且簡潔有力的技術，一個複雜的結果可以透過定義、接連更換部分簡單的初始物件來達到目的所求。其他以字串改寫為基礎的計算模型包含圖厄系統(Thue systems)、迭代函數系統(iterated function systems)、馬爾可夫演算法(Markov algorithms)、標記系統(tag systems)。L-system 屬於一種形式文法(formal grammars)，具備其規則與符號特徵，語法結構上與半圖厄文法(semi-Thue grammars)如 Chomsky grammars 極為相似，通常被定義為如下所示的多元組：

$$G = \{ V, S, \omega, P \}$$

V：變數符號集合(variables)

S：常量符號集合(constants)

ω ：初始狀態串(start, axiom or initiator)

P：產生式規則 (rules)

根據不同的文法 L-system 會產生不同形式語言與各種產生式的規則應用，依照 Manousakis(2006)的文法結構分類，L-system 涵蓋下方所列的多種變化類型：

- Context-free (OL systems) or Context-sensitive (IL systems).
- Deterministic (DL systems) or non-deterministic.
- Bracketed.
- Propagative (PL systems) or Non-Propagative.

- with tables (TL system).
- Parametric .
- with extensions, (EL system). E.g. Multi-set, Environmentally-Sensitive, Open.

這些文法類型在分類架構上並非完全獨立，它們彼此上可以交錯或合併使用，像是 DOL-system 就是指包含了 Deterministic 與 Context-free 的文法特性，可以參考圖 2-25 中所展示出以 Parametric DOL system 生成的結果；圖中可以看到圖像化的規則，左上為植物不同部位的生長與發展順序、左下為對應的生成參數，而圖右便是依據該參數表生成的各種植物外貌。就文法格式上以最簡單的 OL 為例，它的產生式格式可以寫成 predecessor → successor，在圖 2-24 引用原始 L-system 用來生成藻類的模型作為範例說明，比對左邊的系統架構與右側以此系統產生的字串結果，可以發現前面的字元會依據生成規則替換成後面的字元。

Variables: A B

Constants: none

Start: A

Rules: A→AB, B→A

A

AB

ABA

ABAAB

ABAABABA

ABAABABAABAAB

ABAABABAABAABAABAABA

圖 2-24 Algae 範例

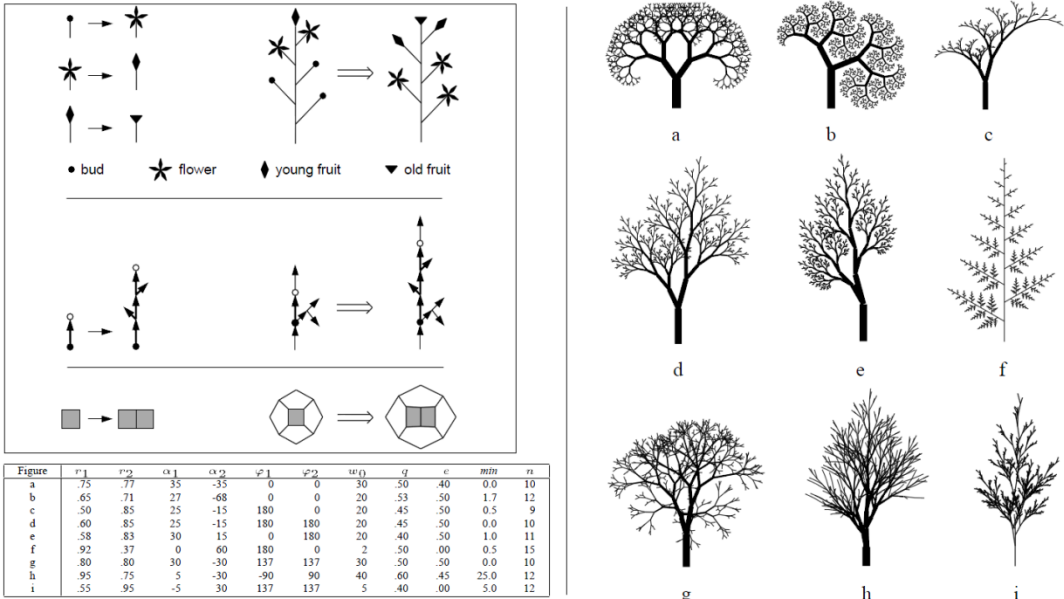


圖 2-25 Parametric DOL system 植物生成範例

在 Müller(2006)與 Martin(2010)等人各自針對建築物與場景生成的研究中，可以看到另一種以 CGA Shape 為基礎、包含 Non-Propagative 與 Parametric，提供可控制與隨機的機制，能根據不同的條件狀況與機率參數來轉換前後對象，其文法格式可寫成 $\langle predecessor \rangle : \langle condition \rangle \rightarrow \langle successor \rangle : \langle probability \rangle$ 。以 Müller 研究中的範例說明，在產生式 $fac(h) : h > 9 \rightarrow floor(h/3) floor(h/3) floor(h/3) : 1.0$ 中，當參數 h 大於 9 時，會有百分之百的機率由 3 個 $floor$ 替換 fac 。

作為程式語言 L-system 和正規文法所產生的語言最大不同處在於，它允許重複平行與平行不連續的規則，亦指 L-system 在一次迭代中可同時套用許多不同的規則，表示說每次導出結果的步驟中，所有字串上的符號不是一個接一個的替換，而是同時作業。這個差異點反應出 L-system 的生物動力(biological motivation)，結果能捕捉到多細胞生物在細胞分裂時同時發生的狀況 (Prusinkiewicz & Lindenmayer, 1990)。L-systems 上的平行生產應用(parallel production application)對改寫系統(rewriting systems)最大的影響在於提供 Chomsky grammars 無法產生的語言。

1968 年，Lindenmayer 學者提出一種新的系統型態用於描述與模擬多細胞生物的生長過程，而後該系統正名為 L-system。L-system 之系統型態與計算機領域上的抽象自動機(abstract automata)和形式語言(formal languages)密切相關，也因此引來學術及業界的高度關注，隨後在科技發展下蓬勃發展。

它的功用在植物模型上尤其顯著，這個優勢於 1984 年 Smith 引進電腦圖學將結構可視化(visualize)並進一步對過程模擬建模後登上高峰。90 年代後已被廣泛應於植物生長的研究，此外還有都市規劃(Parish & Müller, 2001)、建築造景(Marvie, Perret, & Bouatouch, 2005)(Müller, Wonka, Haegler, Ulmer, & Van Gool, 2006)，以及遊戲中的情境或場景生成(Martin et al, 2010)。

通常 L-system 會使用終結符與非終結符(terminal & nonterminal symbol)來指定它推導規則的元素，並搭配龜圖渲染出我們電腦畫面上看到的結果，如圖 2-26。

在Marvie(2005)以L-system變化形態發展出用於渲染建築物外觀的Functinal L-system (FL-system)中，使用不同以往傳統符號的方式，將每個元件定義成不同函式，以函式替代符號作為傳遞參數、生成物件的手段，當這些函式存在參數條件，程序會執行並回傳函式中的參數以創造物件，FL-system除了允許控制規則外，也可以透過函式回傳來改寫並停止推導過程。在圖2-27可以觀察到Martin(2010)簡化的範例，裡面說明一個場景腳本將產生一個訓練目標(TO1)，根據訓練目標要求必要的組件(TARGET、A、OBS)，不同的組件依照各自的函式規則與機率創建對應的實體(artillery、observer、tank、apc)以及針對各自的位置去呼叫下一層函式來選定座標。對於擁有大規模系統架構的生成對象，FL-system與Parametric提供傳統L-system沒有的優點，但在本質上都是位於同樣的基礎；透過遞迴次數L-system可以渲染出漂亮的幾何圖案甚至複雜的植物模擬，我們可以看到L-system有各種不一樣的型態與應用。

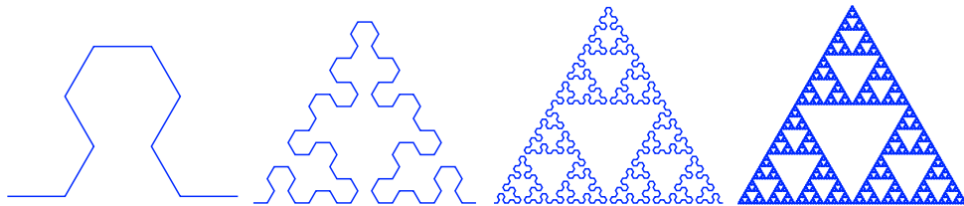


圖2-26 Sierpinski triangle

Rule 1:	{SCENARIO}	:	→ {TO1}	:	1.0
Rule 2:	{TO1}	:	→ {TARGET} {A} {OBS}	:	1.0
Rule 3:	{A}	:	→ {artillery} {POSITION}	:	1.0
Rule 4:	{OBS}	:	→ {observer} {POSITION}	:	1.0
Rule 5:	{TARGET}	:	→ {tank} {POSITION}	:	0.5
Rule 6:	{TARGET}	:	→ {apc} {POSITION}	:	0.5
Rule 7:	{POSITION}	:	→ {position}	:	1.0

圖2-27 Grammar Representation的簡單範例

第五節 小節

透過整理歸納過去分類遊戲的分類方式，解釋遊戲機制、遊戲玩法與特色在分類上的決定性，我們可以統整出平台遊戲的主要構成因子。於分類上平台遊戲以遊戲內部空間佈局裡用來作為玩家駐足活動的各式平台為其主要特色，也因此這類型遊戲挑戰常見於讓玩家在各式平台地形跑跳；例如從那邊的平台跳過懸崖來到這邊的台階上，這使得平台遊戲在人物移動的操作上或多或少包含了動作遊戲需要玩家精確操作技巧的基本要求，使得平台遊戲在關卡設計上須考量到玩家角色一些物理與非物理的運動特性，從另一個層面來說，空間佈局在平台動作遊戲中顯得極為重要。

雖然遊戲基本上都依循遊戲設計的原則製作，但從關卡設計的文獻回顧中，可以看到不同類型的遊戲基於各自的遊戲性質與機制擁有多樣的設計模型，以關卡設計的角度來說，並不存在一個通用於所有遊戲的設計框架，多數的遊戲架構是以單獨類型遊戲分析為基礎。過去針對平台遊戲設計整理出的通用型元件，常見的有構築玩家角色活動空間的平台、敵人角色或死亡陷阱以及類似金幣之類的回饋道具等。

本研究將以功能性層面，意即在達成遊玩目的上的必要性，選用並參考已透過論證且嚴謹的 Smith 提出之針對 2D 平台動作遊戲關卡設計的模型架構做為主要依據。該模型考量到平台動作遊戲在空間布局設計上與遊戲節奏的密切關聯，以遊戲節奏規劃遊戲中每個段落的挑戰，注重遊戲元件實質在關卡裡的功用，在分析面向上偏向功能導向，並且此框架已實際用於關卡編輯器足以證明其完整性，是個理想的關卡架構模型。

在關卡程序生成演算的部分，將結合 L-system 完成自動生成關卡的程序。該系統已廣泛運用於各領域之中，尤其都市規劃、植物生長等較著名的成果，這些應用顯現出 L-system 在空間佈局上良好的表現。本研究也將參考其餘文獻中程序性內容生成遊戲的生成原理，延續其思路在規劃生成過程中確保有路徑解答，注

意關卡設計上應有的邏輯規則，不全權交由系統隨機安排，並嘗試讓系統能藉由更新參數表來學習優化關卡內容。

本研究考量到實作上欲體現關卡生成中難易度參數化的設定，將簡化可能造成難易度變化的變因元件以增加整體程序可行性。在 Smith 提出的原架構中構成挑戰群(Group)的各個元件比例並非固定、甚至可以某幾項等於零，因此不必擔心部分的元件增減會破壞結構平衡，此外，本研究著重點在生成具可玩性的關卡，而非聚焦在加深耐玩性的設計研究上，本篇研究僅取觸發器以外的前四項元件平台、障礙、移動輔助器、收集品構成關卡設計基底模組，而每項元件僅取部分代表，這部分在第三章研究方法中會再度聲明並詳細論述。

本研究目的為提供此作法的雛型；一個較為便捷且可行的程序性生成關卡系統，藉由論證的模型架構輔助 L-system 生成關卡，在未來運用上設計師仍可依自身需求更改部分或擴增加入其他元件。

第三章 研究方法

本研究之目的在於運用 L-system 生成 2D 平台遊戲的關卡，本章將就實作的程序架構與生成方式進行說明，並依序說明邊界容錯和難易度的處理。

第一節 生成方法

本研究以 2D 平台遊戲關卡作為生成對象，並定義每一個關卡擁有一個棋盤網格(grid)，而每一個棋盤由數個房間(room)組合而成，房間之下又有各個樣式的模板規範房間內部的元件；類似於地底尋寶的架構，但不同的在於一開始不會實體化這個代表關卡世界大小的棋盤，以保留些彈性給使用者決定。

本研究將房間定位成遊戲關卡中提供線性玩法的區塊，也就是玩家實際活動的區域，這些房間藉由生成演算法以初始房間往外延伸的方式確保解決路徑(solution path)及串起整個關卡的挑戰動線。因此為了維持關卡內部必要的挑戰，這些房間本身應該要有自己的設計模板，本研究引用 Smith(2008)的模型架構為基礎，選用平台、障礙與收集品等關卡元件規劃房間布局，形成一間間以房間為單位的挑戰群，並提供使用者一套基本的房間樣板。

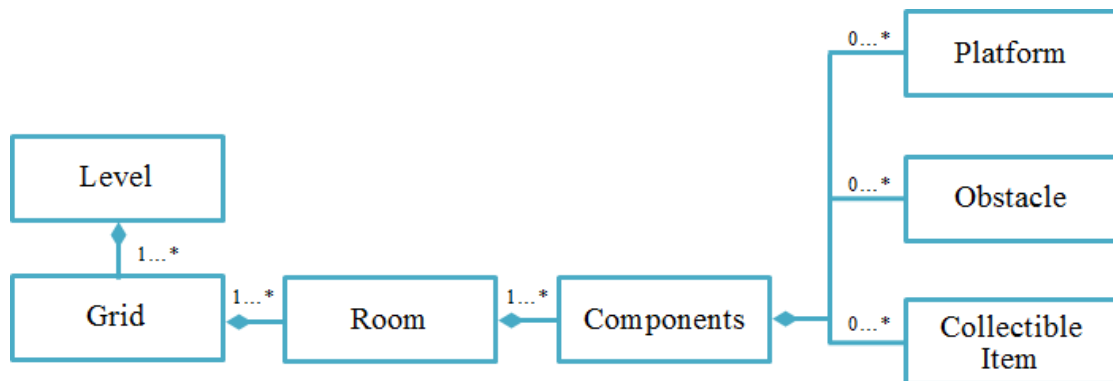


圖 3-1 架構概念圖

本研究採用 FL-system 以函式代替符號進行生成的手段，並具備以參數控制部分生長的規則及機率，整體流程可參考圖 3-2 與搭配表 3-1 的語法規則；本研究生成系統在開始進行生成前會先讀取預設值與使用者設定，再依據容錯處理的

結果決定是否覆蓋掉預設的系統參數，緊接著實體化棋盤網格和初始房間，做好生成前的準備後，便正式進行 L-system 的生成程序並同步輸出結果。

關卡產生的遞迴過程如表 3-1 所示；根據存放房間資訊的陣列 roomPool 裡有無可參考的房間對象，生成對應的子房間，接著依要求生成子房間的指令檢查確認是否可以生成，如果可以生成，也就是 $\text{type} < -1$ （參考表 3-2 的說明）則再進一步檢查它的周遭，並依據周遭環境剔除不可行的房間樣式，然後隨機選擇房間樣式並產生房間實體，產生實體後再度把房間資訊丟入 roomPool 為下次生成做準備，如此重複循環系統便能產生出最終要求的關卡地圖。表 3-1 中所使用的函式概念名稱可參考表 3-2 的定義。

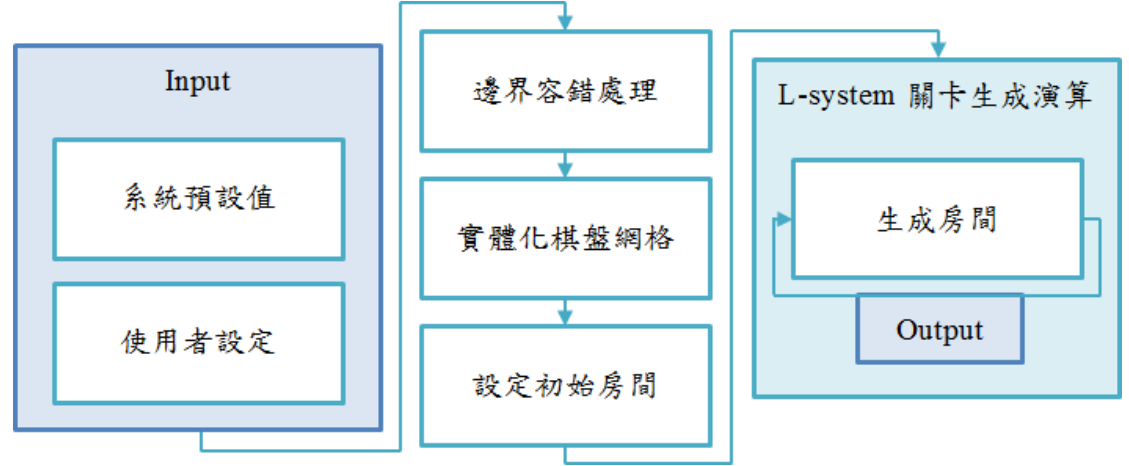


圖 3-2 系統流程圖

表 3-1 關卡生成的遞迴規則

1 : {CreateRoom} :	→	{CreateSubRoom} : 1.0
roomPool.Pop()		
2 : {CreateSubRoom} :	→	{Check} : 1.0
3 : {Check} : room.type < -1	→	{CheckSurrounding} : 1.0
4 : {CheckSurrounding} :	→	{SelectRoomType} : 1.0
5 : {SelectRoomType} :	→	{CreateRoomEntity} : 1.0

6 : {CreateRoomEntity} :	→ roomPool.Push(room){CreateRoom} :
	1.0

表 3-2 關卡生成規則中的概念表

符號與函式概念	定義
CreateRoom	開始進行內部房間生成的準備程序；用於定義和產生 L-system 疊代生成時所需的初始值與相關參數。
CreateSubRoom	開始進行周遭房間生成，並以指定房間做為參考的父房間。
Check	確認棋盤上的空間，確定是否可以於該網格中產生新的房間。
CheckSurrounding	確認將產生的房間周遭的其餘房間樣式，包含相鄰面是否為棋盤邊界與挖空的區塊。
SelectRoomType	依照周圍鄰居檢查結果與隨機亂數選定房間樣式。
CreateRoomEntity	產生房間實體物件，並呼叫內部函式生成台階、障礙、收集品等內部元件。
RoomPush	執行 roomPool.Push(room)的函式指令。
room	指生成程序中構成關卡的房間。
type	紀錄房間樣式的值，因樣式種類標記只用到零與正整數，因此將-1 定義為挖空的標記、-2 定義為無樣式的標記，也就是該房間沒有樣式。所有棋盤上的房間在生成前皆為無樣

	式的空房間，因此可藉由確認 type 值是否 小於-1 來得知該空位是否可產生新房間。
roomPool	指存放房間資訊的陣列

與其他程序性生成關卡相比，本研究並非以單一解決路徑作為房間布局的策略。就程序性上來說，本研究在一開始就不是以起終點的路徑作為生成路線，而是單對每間房間的連貫性，以起始房間為中心探索能抵達的區域來做為生成手段，也因此以本研究方法產生的關卡在沒有額外使用微調設定前，是一個以起點為中心的網狀散射地圖。

雖然一般的遊戲設計在關卡中會同時存在起點與終點，但在這個關卡生成程序中只有固定的起點，終點則交由程序性隨機產生或讓使用者自行決定。在程序定義上，遊戲起點位於作為 L-system 生成程序中初始房間，稱為 axiom，而當生成系統結束所有的關卡路線，系統會在最後一輪生成的房間中放入終點。

考量到此關卡生成系統產生的地圖隨著遞迴次數增加可能變得極為寬廣，另外起點的位置從中心到角落都有可能，系統在預設上建議設置多個終點，原則上會從最後一輪的房間群中挑選一半的房間設置，假使使用者有自己的需求也可再進行微調或者於生成結束後再從眾多房間中挑選合適的位置做為出口。

另外關於關卡生成內不同房間的定義，根據這些房間作為遊戲關卡中的活動場域可以水平擴展與往上或下延伸，本研究中的房間具有 15 種樣式(type)，這裡的樣式並非指花樣、風格等美術設計，而是依據通道的出入限制所做出的基本規範，並於程序中標上連貫的 0~14 編碼，如圖 3-3 所列舉的樣式示意圖；圖中橘色方框中的黃色區塊代表每間房間實際提供給玩家活動的空間，而每處以膚色線段標記黃色區域與橘色房間外框邊緣重疊的地方為出入其他房間的連接口。

須注意圖中的樣式外觀僅只是用於視覺化的表達，在樣式的定義上同一類樣式房間可能會有許多種面貌，端看每個房間樣式所擁有的樣板而定，所為根據房間樣板也就是指由規範內部物件的參數表決定。其中考慮到在未來擴充上，房間不一定使用磚(tile)為構成單位，而有可能分區域來設計場景物件，因此排除使用鏡射或旋轉的方式簡化樣式種類。

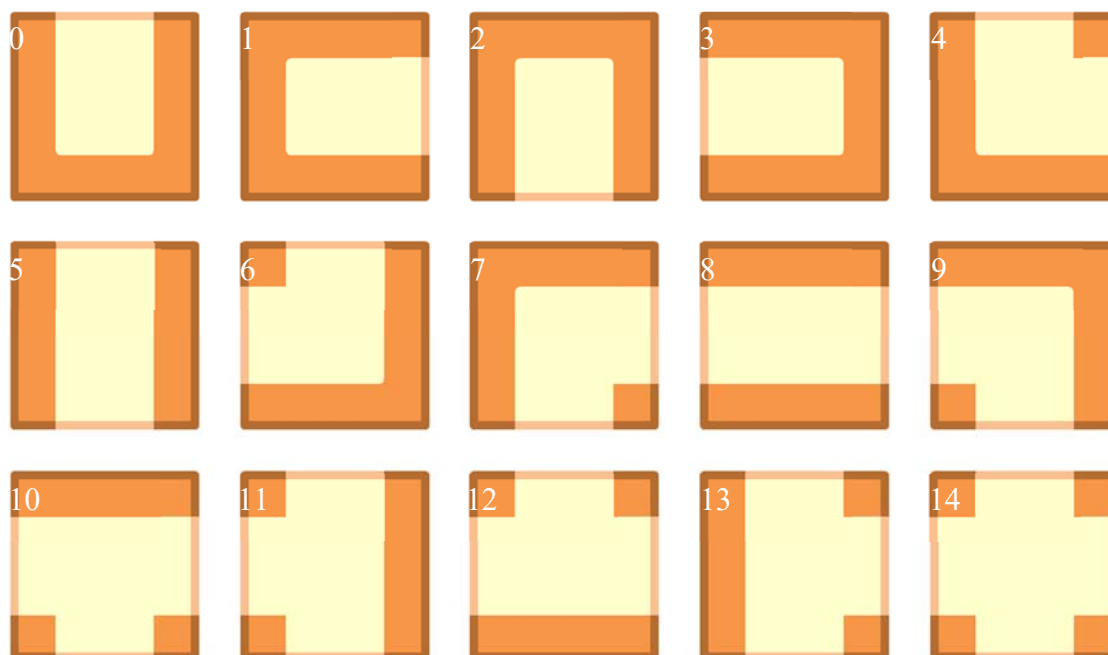


圖 3-3 14 種房間樣式

第二節 邊界容錯

針對邊界處理的問題必須先回到關卡空間的規劃上；在基本的空間宣告裡，依據每次遞迴程序上會往周遭再延伸一間房間，會預先保留適當的空位以確保所有生長的可能，假使棋盤大小固定是 $n \times m$ 個網格，因為以初始房間為起點向四面延伸最遠的距離直接等於遞迴次數，我們至少需要 $(2 \times \text{遞迴次數} + 1)^2$ 大的棋盤才能容下全部的房間，但同時也知道實際上程序最多會生成的房間數量只有 $2(\text{遞迴次數})^2 + 2 \times \text{遞迴次數} + 1$ ，如圖 3-4 所示。

以一個簡單的例子為例，假定初始值為位於棋盤正中央、四面都能往外連通的房間且遞迴次數為 1，系統會預設棋盤為 3×3 的網格，然而因為每次遞迴只會

向周圍可連接的區域執行一次生成程序，加上最初的起始房間實際被生成的房間數量為 5，僅占用棋盤約略一半的空間。儘管如此，並不可能以較後生成的房間物件決定先定義的棋盤甚至不規則的邊界外型，棋盤在整個生成程序中也只是代表空間的二維陣列並不具有實體。

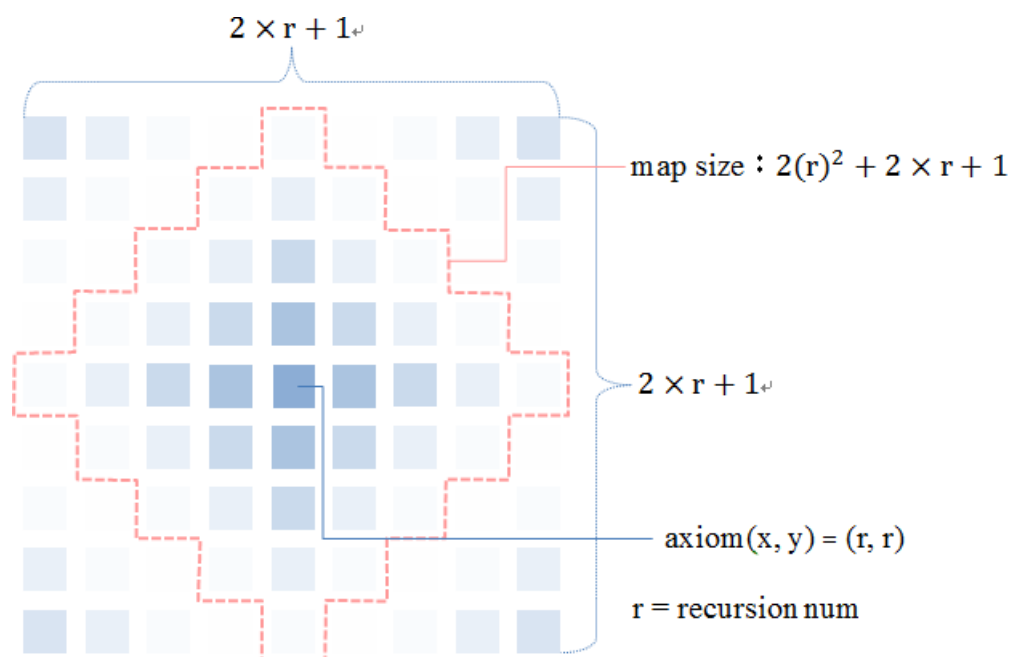


圖 3-4 關卡地圖預定空間

為了解決空間上的浪費，本演算法提供使用者設定棋盤邊界與初始房間位置，讓每一關的棋盤大小除了根據遞迴次數也會依使用者設定作出調整，並且增加使用者可預先挖空棋盤區塊的設計，如圖 3-5 的示意圖說明。

圖中以有邊線的綠色、藍色、黃色方框依序代表依遞迴限制次數產生的預設棋盤、使用者設定的棋盤以及最終系統藉由交集計算出實際棋盤外貌，此外以較小的墨綠、深藍、橘色方塊標記這些棋盤內的初始房間，並做為交集演算時的中心點，而黑色與白色的小矩形則表示使用者預想挖掉的棋盤區塊與驗算後確定在最終棋盤內並實際挖掉的區域。須注意本圖為棋盤預視化的想像，在整個生成程序中棋盤僅只是個陣列。

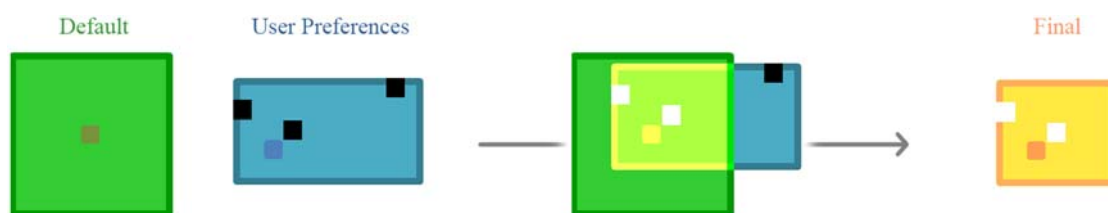


圖 3-5 根據預設值與使用者設定的交集決定實際棋盤大小

在開始生成關卡前，系統將再次確認邊界、起始位置與房間樣式彼此的關聯，以避免錯誤產生不恰當的結果；當使用者的設定自相矛盾時，系統會採取替換的手段將起始位置依照邊界範圍作出變更，以及參照起始位置對起始房間樣式編碼進行更動，以圖 3-6 中例子來說，系統會將超出範圍的初始位置設定移置棋盤邊界內最近位置，並修正房間的樣式編碼使後續房間能正常產生在棋盤內，如果位處等距的狀況，例如圖 3-7 中系統會依順時鐘的方式挑選鄰近位置。

在這些使用者設定錯誤的情況中，系統以優先權順序先處理位置在考慮樣式的問題，也因此系統對於產生房間初始樣式上會優先考量有較多擴展可能的房間樣式，而只有在使用者設定沒有問題時系統才會採納使用者設定，這樣的狀況可以在圖 3-5 與圖 3-6 中看見。另外，假使因為挖空棋盤的關係，如圖 3-8 中房間落在孤島上，系統會判定使用者只希望產生一個房間，而生成 1 個僅有一間房間的 1×1 棋盤。

整個程序大致依循“檢查起始房間是否超出邊界”→“如果是則修正位置至邊界內”→“檢查起始房間是位於棋盤上挖空的空洞內”→“如果是則順時針訪問周遭區域以修正位置至空洞外”，如果使用者把棋盤上所有區域都挖空或只留下一格則跳過所有檢查步驟。

在示意圖 3-6~圖 3-8 中，統一以白色網格代表棋盤大小，灰黑色代表挖空的棋盤區域，橘色方框代表一個房間，橘色矩形內的黃色區塊則代表房間內可活動的空間，該區塊貼合底部邊線的部分則是連往其他房間的出入口，以圖 3-6 從左到右來說，首先將房間移進棋盤，接著將房間樣式從朝上的出口改為向右和下。

而圖 3-7 與圖 3-8 個別只有一個步驟，分別為把房間從灰黑色挖空區域移置白色棋盤上與根據房間被孤立的情況將白色網格縮減。

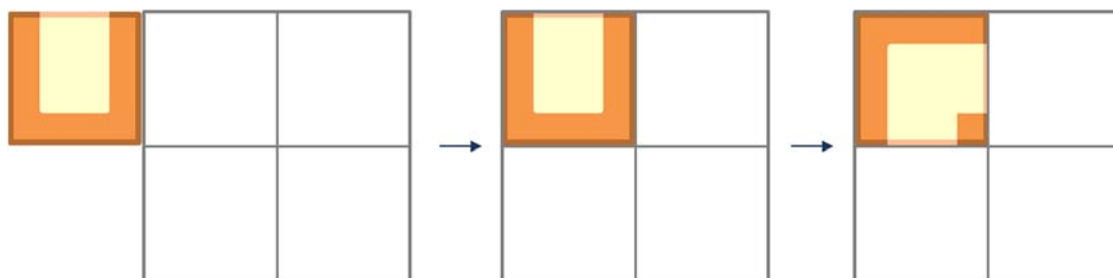


圖 3-6 使用者錯誤設定在邊界外的範例

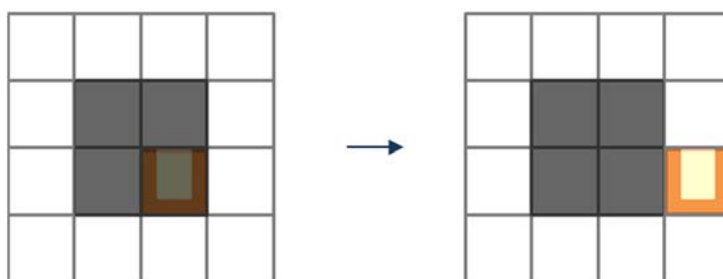


圖 3-7 使用者錯誤設定在挖空區域內的範例

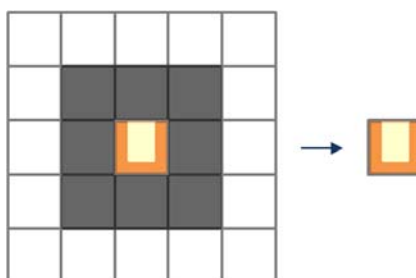


圖 3-8 使用者錯誤設定的孤島範例

與此相對的，房間數量則會依據遞迴生長狀況以及棋盤邊界限制作出變化，並為了體現多樣的生長狀況，不會強制生成的房間填滿所有空間，這也是因為不希望生成一個四通八達沒特色的地圖，因此本研究所提出的生成機制在填滿棋盤的過程中並不會頻繁選用通道較多的房間，而採取所有樣式機率均等以保持產出的關卡地圖隨機多變的可能。本研究的生成機制唯一排除掉在最後一輪前的房間生成不會採用圖 3-3 中 0~3 的樣式，亦指只有在房間路徑的末梢位置才可能選用

一端開口的封閉房間，這是為了避免關卡地圖過早封閉，但因應使用者需求這個機率是可以修正的。

第三節 難易度處理

如同房間的功能是為了提供遊戲挑戰，本研究將關卡的難易度變化設定在房間內部構成上，並以模組化的方式規範這些變化的限制。本研究將 L-system 架構用來生成遊戲關卡的所有房間位置，亦即遊戲的主要地圖，而關於房間內部的構成環境，則交由每個房間各自處理。

在實踐上，本研究讓每一個房間擁有自己的樣板(template)，意指參數設定表，這其中包含一些可變動的布局，像是道路規劃、敵人或陷阱物件的擺放，當房間被創建時，每間房間會根據各自的參數表生成房間內部元件，關卡難易度便是利用參數對這些物件做細部調整。

障礙物是所有功能元件中最常用作控制難度的物件，在這些物件生成設計上，本研究考量到玩家在房間中遇到的障礙越多，代表該房間難易度越高，但因為每間房間受樣式種類影響會有 1~4 種通道開口，如果讓系統隨機取樣房間內的座標位置，玩家不一定能感受到難易度變化帶來的差異。

以圖 3-9 的 T 字型房間為例，玩家在進出房間時可能不會經過房間內所有道路，導致有時會忽略或無法觀察到房間內已設置好的障礙與挑戰，並且就算路線如何規劃，對玩家而言始終不存在把整個房間都逛過的必要，遊戲設計上也不該有這樣的硬性規定；這當關卡地圖越長越大時會有其他問題產生。

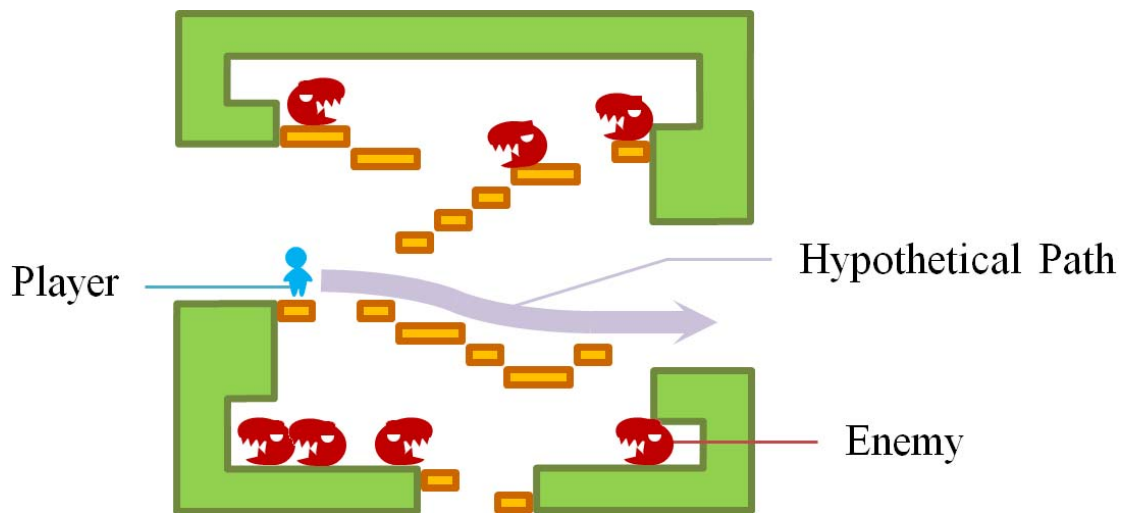


圖 3- 9T 字型房間範例圖

因此，本研究提供使用者針對房間樣板做細部的設定，例如規劃台階路線、障礙物件生長區域、物件生成期望等，雖然仍可能發生些許不如預期的情況，但跟程序系統相比，使用者較能站在玩家的角度對遊戲空間進行布局。本研究建議使用者可依需求指定定義函式，或採用本研究所提供的預設參數表將這些物件納入演算。

為了實作上的需求，本研究固定目前房間模板內會使用到的物件種類，然而這連帶導致在難易度處理的設定上會產生無法預期的狀況：如是一些難易度的鋪成與元件上的設計會因為遊戲本身擁有的元件種類與數量有所限制；例如在沒有爬梯的狀況下，關卡中的垂直路徑該如何布局，又例如在既有元件下關卡難易度該如何體現。但為了解決這些問題也不宜無限上綱物件種類，因為這是無論有多少種類物件都是必須考慮的事，本研究建議使用者針對實際需求選擇相應的物件即可。因此未來使用者若想擴充多餘的物件種類，本研究建議可重新編寫相關的邏輯程序，以達到較好的遊戲性。

作為讓使用者設定難易度參考的基礎，本研究目前將房間難易度預設分為從 0 到 4 的 5 階段，並限制難易度僅止影響房間內四種物件，分別為台階、敵人、陷阱以及光球；此處的光球指作為收集品的回饋道具，類似瑪利歐遊戲中金幣的概念。以下將針對這些物件個別說明難易度的處理：

一、 台階難易度設定

本研究針對台階設定在每間房間中擁有數條由台階組成的道路，這些道路於模板內各自定義了固定的起終點，而難易度參數便是控制這條路經上台階會如何擺置。為了方便控管以及確保生成的台階路徑在遊戲中具備可行性，本研究首先依據玩家角色尺寸與對應的玩家角色能力極限定義台階的長寬。

本研究假定遊戲世界中最小的單位量為 a ，而玩家角色佔據 $a \times 4a$ 的大小，其基本奔跑跳躍時的水平移動最大值為 $4a$ 、垂直跳躍最大值則為角色高度 $+a$ ，並定義台階的寬度不可小於角色寬度，台階彼此間距則必須控制在水平移動最大值的 $4a$ 內，只有在垂直排列上允許同垂直跳躍最大值的 $5a$ 間隔，雖然在程序中因為角色與台階的碰撞具有選擇性穿透，某個層面可把台階視為一條線段，但建議視覺上的貼圖長寬最好大於 a ，讓玩家有實質踩在物體上的感受。

本研究考量到在範例關卡設計上並無爬梯或傳送門等便利的物件，在一些高度差過大的路經上台階水平面的數量必然遠小於垂直面，因此本研究設定台階本身具有穿透性，亦即玩家可透過垂直跳躍從下層台階跑到上層，並且可以透過落下的按鍵操作，從上層台階跌落至下方的台階上。

在本研究範例中提供 $4a \times a$ 與 $2a \times a$ 兩種台階規格，並以最小單位量 a 做為規範台階與台階間距離的單位限制。根據此最小單位量，本研究定義台階間的水平間距可落在 $[0, a, 2a, 3a, 4a]$ 此區間範圍中，並且不可超過玩家角色在奔跑跳躍時的水平移動最大值 $4a$ ，而垂直間距在平台間無垂直面交集上可同樣落在該區間內，反之，假使兩個平台位於同垂直面上，則為了確保之間的縫隙不致卡住玩家角色且玩家

能控制角色跳上較高的一側，平台間高度差限制只能有 $4a$ 與 $5a$ 兩種可能，也等同玩家的角色高度與最大垂直跳躍距離。

本研究僅提供使用者設定台階路徑的起始與終點位置，並規定所有路徑上的台階由左到右依序生長。使用者在設定時需盡可能避免交錯的道路，假使希望規劃如 X 交錯的路，最好拆成四段路徑，而為了避免影響路線的垂直面，在台階的難易度控制上只針對台階間水平的間距做變化，階梯的垂直變化則依據起終點兩端的高度差產生連動影響。例如當兩邊的高度差很小，那麼可能採取間距 $0\sim a$ 作為水平位移後的垂直變化值，反之可能大量選用 $4a$ 的間距。

假使生成最後一階台階時如示意圖 3-10 中發現離終點的垂直高度差大於 $5a$ ，則依狀況微調最後一階平台，或於其上或下方區域以 $4a$ 的間距間格生成連往終點的台階。對於台階垂直生長的限制除了防止不同路徑交互干擾，也希望這樣產生的階梯能確實伸展到預期的終點。

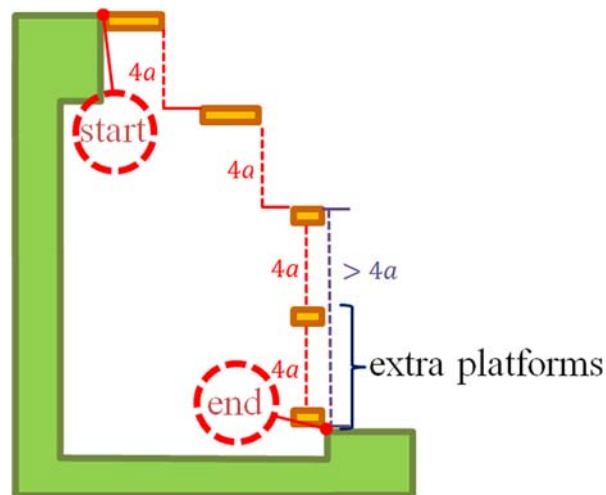


圖 3- 10 額外產生台階示意

本研究設定不同的難易度對應不同的間距期望，可參考表 3-3，本研究考量隨著間距約大難易度會正向上升，但如果每個難度只對應一種間距，例如難易度 4 就對應間距 $4a$ 這樣似乎太過單調，且在未來擴

充上會有較大的限制，因此採取機率的形式，在生成一段階梯道路時，會透過這個機率表決定路線上台階可能的間距，例如以難度 2 生成的路徑可能包含了 3 種不一樣的間距，如圖 3-11 的示意圖。這些平台間距期望機率可由使用者自行微調，建議上以最高難度包含所有間距、最低難度只能零間距，其餘的部分依固定比例調整。

表 3-3 平台間距的生成機率表

難易度	間距 0	間距 a	間距 $2a$	間距 $3a$	間距 $4a$
0	1	0	0	0	0
1	0.5	0.5	0	0	0
2	0.25	0.25	0.5	0	0
3	0.125	0.125	0.25	0.5	0
4	0.0625	0.0625	0.125	0.25	0.5

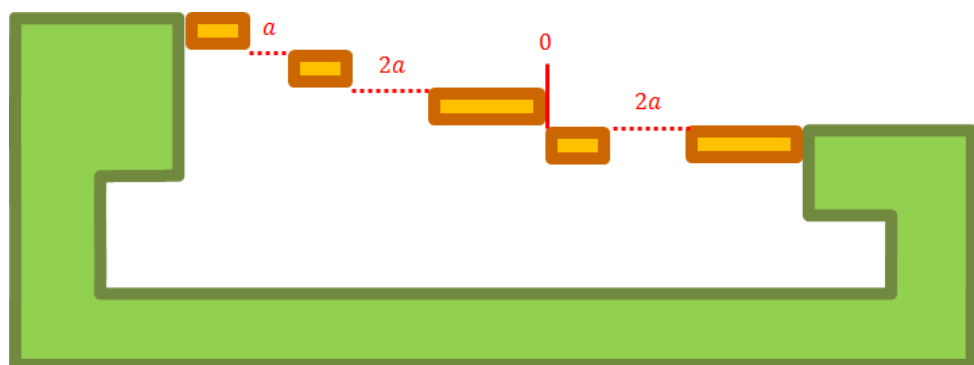


圖 3-11 台階難易度 2 的生成示意圖

二、 敵人難易度設定

本研究將房間內部空間劃分為數個區塊作為敵人角色生成的參考位置，根據不同的樣式模板，這些區塊的大小比例與位置各不相同，由使用者在系統生成前定義出期望可遭遇敵人的區域，假使關卡中只使用到不會飛的敵人，那麼建議使用線段代替區塊定義期望範圍。在未來這些區塊可整合成一個不規則外型的參考範圍，但目前由數個矩形表示。

本系統定義出一個敵人數量期望表，並且以區間範圍增加數量變化，如表 3-4 所列，當難易度越高相對期望產生的敵人數量越多，以難易度 2 來說，其對應產生的房間內敵人可能會有 4~6 個。當系統開始進行生成時會藉由難易度參數決定生成的敵人數量，然後讀取期望生成的區域以及生成的機率，最後根據這些值產生畫面上玩家看到的敵人角色群。表中期望區間的數值可依使用者的需求更改，建議上區間彼此不宜重疊不然可能無法實質體會出難易度的變化。

表 3-4 敵人生成數量與難易度對應表

難易度	0	1	2	3	4
生成的敵人數	0 到 2	2 到 4	4 到 6	6 到 8	8 到 10

整個過程可參考示意圖 3-12~3-13，假定圖中三個指定區塊分別被選中的機率 p 為 0.5、0.3、0.2，系統每次產生敵人物件會依此機率挑選生長區域，然後從該區塊範圍內隨機取樣一個位置參數作為敵人物件的生長點，並將已挑出的生長點暫存至陣列中，當後續取樣時發現生長點重複或距離過近可能會有物件重疊的情況時，便能將該生長點依據敵人物件寬度做水平平移，如果左右移動後發現都會超出指定的區域範圍，那麼重新選擇其他的區域，假使不存在合適的區域那麼便

停止產生敵人；如果使用者發現房間內的敵人數量遠少於預期，那麼可能得調整下敵人生長區域，確認是否是區塊設定太小。

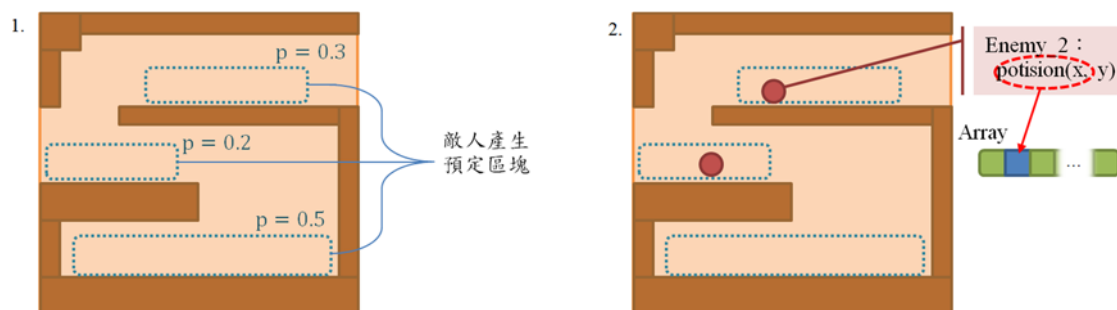
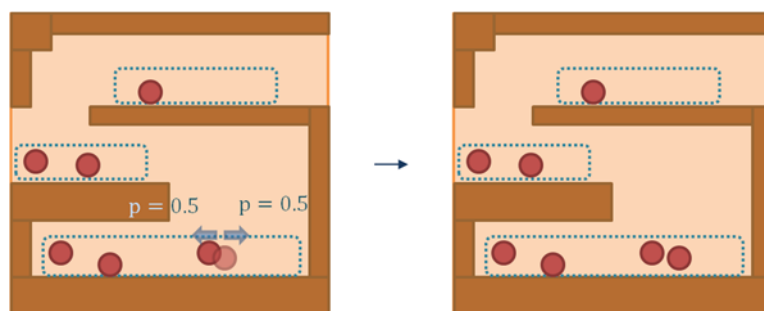
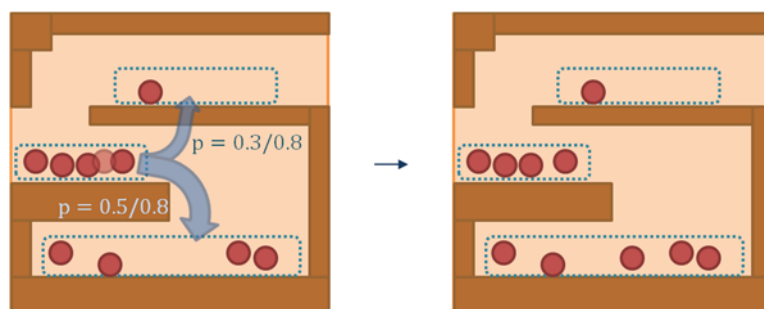


圖 3-12 敵人物件生成步驟



(a)指定區塊內敵人物件數量較少時以平移處理重疊問題，預設左移或右移的機率各為 0.5



(b) 指定區塊內敵人物件已滿時以更換區塊處理重疊問題，設定機率為原先機率除以餘下的機率和

圖 3-13 敵人物件重疊處理

三、 陷阱難易度設定

陷阱與敵人的難易度設計大致雷同，只是陷阱本身為靜態、被動的物件，通常其位置在產生後便固定在地面，例如市面遊戲中常見的尖刺、泥沼或瑪利歐裡會咬人的植物，在本研究範例中採用同樣只能立於地面上、不可浮空的陷阱物件，因此改成全面以線段的方式取代敵人以區塊的方式劃分可能生長的範圍。

使用者在定義時並須格外注意座標點位於地面，假使額外需要浮空的陷阱物件則建議最好另外設置，不宜共用同樣的參考線段作為物件產生點的座標取樣範圍。與此相對敵人因為是可活動的物件，座標設定上一些微小的誤差在遊戲開始後會因為動態移動而掩蓋，使用上較陷阱自由些，使用者在選擇房間內布局的物件時建議將不同種類物件的特性考慮進去。

這邊也同樣定義陷阱有類似敵人物件的期望表在表 3-5，隨著難易度越來越高，陷阱數量也會越來越多，表中的預設期望值也同樣可以依使用者需求更改。

表 3-5 陷阱生成數量與難易度對應表

難易度	0	1	2	3	4
期望區間	0 到 1	2 到 3	4 到 5	6 到 7	8 到 9

四、光球難易度設定

在制定光球難易度前，必須先定義作為收集品的光球在遊戲中的定位是甚麼，以及每一關存在的光球數量是否要固定或不固定，假使採用類似鑰匙物件需要收集齊全才能通關的特性，往往會使用固定數量，並採取隨機或由設計師規劃藏匿的位置，這對於本研究生成的關卡並不合適，尤其當關卡地圖擴大到一個程度後實在很難想像玩家還

能夠堅持下去，因此本研究定義光球性質類似於金幣，用於讓玩家獲得遊戲積分、提高遊戲性與增加成就感的回饋道具。

這類型物件在遊戲中的數量通常是不固定的，隨著關卡地圖越大、可能放置的數量越多，也因為具有可獲得遊戲上獎勵的誘因，玩家會主動去追尋，在關卡設計上也常將它們當作路線引導或者擺放在較危險的地方，誘使玩家自發性挑戰遊戲中的障礙。

這使得難易度變化對收集品造成的影響，常是因為與其他物件相互牽連而發生，也因此光球的難易度控管上較前面的三項物件困難，本研究建議將會跟其他物件牽連的情況合併在與之牽連物件的生成程序上，例如台階路徑上的光球可同時與平台一塊生成，在其他獨立的狀況下則建議依據難易度個別規劃光球在房間中的位置並於生成時透過參數表讀取座標群。

關於整體房間難易度的設定，在預設上統一同關卡內的所有房間共用同個難度，但一樣提供使用者自定義的部分，可設定依照 $0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ 漸漸遞增難度，或者 $0 \rightarrow 1 \rightarrow 2 \rightarrow 1 \rightarrow 0$ 之類的循環變化，或者混用不同物件的難易度設定，例如台階路線以難易度 0 生成，而敵人和陷阱選用 3 或 4 的難度，這種做法適合設計在有新的事物需要學習的教學關卡，讓玩家在簡單的環境下熟悉遊戲內新出現的敵人挑戰。目前程序中針對難易度僅設有 5 個，用於體現難易度上明顯的變化，在未來擴充上也可細分成更小的單位。

第四章 實驗結果

本研究成果在於實際生成一個具可玩性的 2D 平台遊戲關卡，本章將就實作的物件設定與參數定義作為範例展示，另外，為了視覺化遊戲關卡內部的設計，範例中會使用一組固定的關卡元件以體現程序性生成結果；這些元件在往後程序運用上可自由替換成使用者自行設計的物件。

第一節 實驗定義

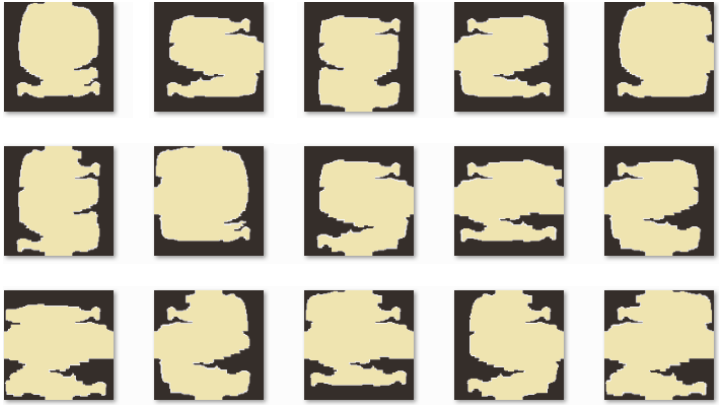


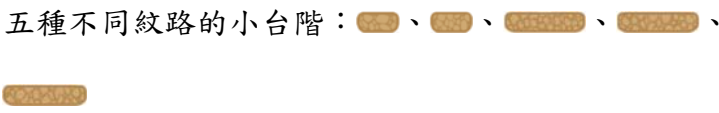



在 Smith 的關卡設計概念模型中，一款平台遊戲設計上基本會包含平台、障礙、移動輔助器、收集品與觸發器五種功能元件，這些元件在關卡中佔的比例並非固定，也不存在硬性規定每個平台遊戲必須包含多少功能元件，它們僅是整理自市面上 2D 平台遊戲的通用代表元件，設計師可隨實際用途增減使用。

在本研究中，將以個別物件產生的方式生成平台和障礙，並作為遊戲中主要與唯一的難易度變化控制，本研究並不打算在遊戲場景中建置爬梯、彈簧床或傳送裝置等移動輔助器，也不考慮放置可能對遊戲環境產生影響的觸發器，這些元件較適合以群組的方式規劃生成，雖然在不影響難易度參數對其他元件的控制之下，也可以固定位置設置移動輔助器與觸發器於房間模板上，這兩種功能元件也可有效提升遊戲關卡的遊戲性，但在展示程序性生成效果的實驗範例中並非必要，本研究在範例遊戲中也並沒有放入攀爬或傳送的機制以及額外撰寫給角色的觸發事件，這些事件也可能複雜化難易度的判定。

目前的範例遊戲中僅提供玩家角色從起點到終點的一段線性玩法，玩家可以操控遊戲角色做基本的跑跳移動，在行進路線上可能會遭遇敵人或陷阱這兩種障礙，以及可收集做為回饋道具的光球。這些範例上會使用到的物件群將統一系列在表 4-1。關於這些物件圖像在遊戲中的比例，本研究參考市面上平台遊戲的人物高度與畫面長度比普遍落在 1：10~1：8 中間，選擇以 1：8 比例設計個物件比，並訂定房間範圍至少為四個畫面大小的方形區域，例如在 1024 × 768 視窗畫面中

人物的高度為 128、房間大小至少為 2048×2048 ，假使房間內會安排放置些佔地較大的隔板物件，建議最好在預留玩家可活動區域下增加房間的大小。

表 4-1 房間內的關卡設計元件

功能元件	物件種類	範例遊戲中使用的實際物件圖像
平台	surfaces	<p>作為底層的 15 種通道背景：</p>  <p>與三種作為牆壁和柱子物件：</p> 
	loops	在遊戲中會上下或左右移動的台階 
	tops	五種不同紋路的小台階： 
障礙	敵人角色	會移動攻擊玩家的敵人： 
	死亡陷阱	可能造成玩家死亡的靜態陷阱： 
收集品	回饋道具	光球  ，類似於瑪利歐遊戲中的金幣

表中將平台分為 surfaces、loops、tops 三塊，它們的定義如 Smith 架構模型中所提出；分別代表整個物件表面都作為平台使用的 surfaces、會依固定路徑循環移動的 loops 與只使用物件頂部作為平台的 tops，在這邊將 surfaces 元件固定為房間模板裡的背景，它們除了系統生成前可進行修改外並沒有多餘的參數控制，

屬於房間模板中固定不變的物件，某個層面來說代表了初略的樣式視覺外觀；房間樣式基本可擁有數種模板，但一個模板只會對應一種背景組合。

本研究生成範例中僅對每種樣式提供一種模板，因此只設計了 15 種背景通道，而為了重複利用物件圖像因此將部分牆柱獨立出來，但它們都屬於通道背景組成物件的一環。建議使用者在未來增添同個房間模板下的新背景元件與貼圖時可統一將物件群組並鑲嵌於 surfaces 平台中。

Loops 就定義上是指會沿著循環軌道來回移動的平台，每間房間可能會有零到數個不等的移動台階，為了簡化難易度處理的規範標準，在生成範例中僅使用單純垂直與水平移動的移動平台，並且事前做好路徑規劃讓移動平台作為固定物件直接綁在背景物件群中，以避免干擾依難易度程序生成的平台階梯路徑。

而 tops 平台元件泛指用於生成房間內部階梯道路的台階，通常房間內部都會有數條由台階組成的道路，這些道路會隨難易度產生變化，在生成範例中本研究限制每間房間模板擁有的道路數量以及路線固定，這是因為每間房間出入路徑是由內部所有道路連接而成，在路徑規劃上必須考慮整體房間的空間布局，假使希望同一樣式房間有不同的路徑呈現，須再模板中額外增加幾組新的道路規劃，就目前的範例裡只提供每間房間模組一種道路規劃，以清楚比較難易度對台階路徑的影響。

在障礙的部分包含敵人與陷阱這些外在的挑戰，雖然在 Smith 的模型架構中平台間距也算障礙中隱性的元件，但避免混淆沒列入表中，每間房間因應模板設計與難易度變化可能出現零到數個不等的障礙挑戰，在生成範例中本研究劃分每間房間中的固定區塊做為障礙生成的參考位置，並定義在房間模板中。

最後關於收集品設定，本研究在目前的生成範例中只放入作為回饋道具的金球，類似於瑪利歐遊戲中的金幣帶給玩家獲取分數的成就，它也是作為難易度影響項目的最後一環，在範例中與障礙物的設定一樣使用模板中定義的固定區塊做為參考點。

接著針對房間外部與生成程序中的定義，範例遊戲中定義初始狀態(axiom)為一個房間，該初始房間編碼預設為代表十字通路的 14，使用者可以在決定邊界範圍、起始位置時一同更改樣式編碼，在生成前系統也會再次確認邊界、起始位置與房間樣式彼此的關聯，以避免錯誤產生不恰當的結果。作為邊界處理上的參考中心點，初始房間預定座標為(0,0)，而後生成的房間則根據相對位置產生在它的四周。另外考量記憶體資源與算圖的問題，限制一次產生的關卡地圖不得超過 1000 個房間，亦即在棋盤大小 31×31 之下的 L-system 遞迴次數最大限制為 15，基本上玩家也不太可能有耐心遊走完如此龐大的地圖。假使整個遊戲世界需要以超過這個範圍的房間數量展示，那麼建議將世界地圖拆解成多個小關卡地圖，玩家也比較能夠消化。

在房間樣版的設計上，雖然沒有明確顯現在生成的結果中，但其實有很大的通道限制問題，目前樣板採用的底層背景通道平台，根據基本定義中的 15 種房間樣式，將房間通道都設計位於邊界線上的中央，這是為了統一通道出入口所在的位置而做出的決斷。假使設計師需要設置房間出入口在某些特定位置，那麼相對地必須針對此一變動增加許多衍伸的樣式樣板。

房間每多一種不同的出入通道，就必須為了所有可能連通的房間，添加八種新的樣板，例如在房間往上的樣式 0 中增添出入口偏向右上的可能，那麼除了樣式 0 也必須為可與樣式 0 連接的樣式 4、5、6、11、12、13、14 定義新的樣板，除非使用者選擇當作例外狀況，那只要為樣式 0 與另外一個期望連接的樣式設計新的樣板。某個層面來說，這也因為本研究不想以 tile 作為構築房間的唯一手段，為了保留手工繪製背景及元件的選項，因此沒辦法以程序化的方式對通道出入口進行調整。

第二節 生成範例

透過 L-system，可以無限拓展關卡內的地圖，在圖 4-1 可以看到基本的疊代結果，這邊將依序說明遞迴次數(n)由 0 到 4 的個別生成程序執行狀況，相關的關卡生成功能函式可參考前面的表 3-1。

在 $n = 0$ 時，只有產生作為 axiom 的房間，因此使用到一次同為後續房間生成使用的功能函式 {CreateRoom}。在 $n = 1$ 與其上的遞迴狀況時，首先必須說明依據規則表面計算出的結果字串似乎永遠只會擁有一個 {CreateRoom}，但實際上在動態規則更動之際，系統便以這些規則導入的函式生成了相應的房間群。

於第二章的文獻回顧中有提及，FL-system 與傳統 L-system 在構成上有些微的不同，雖然它也可以用於產生大量的符號字串，但因為它擁有條件參數與用功能函式代替符號的特性，更適合用來做這種系統化並層層疊加的多功能指令演算。由本研究的實例來說，在創建房間後便會檢查是否可生成子房間，而每次創建完子房間後便又會回到一開頭的檢查創建的房間是否需要生成子房間，因此在表現程序結果的字串上始終停留在固定的幾個功能函式，但在演算過程中，隨著生成規則逐條被導入，構成關卡地圖的房間群也逐間被產生出來。

以過程來說， $n = 1$ 時執行了五次 {CreateRoom}，因為從 roomPool 提取出的資料告訴它需要製造 4 個子房間，雖然子房間的生成手段基本上不管數量多少都包在 {CreateSubRoom} 中一次解決，但在程序中 {CreateSubRoom} 其實便是根據有多少個子房間要被產生再去呼叫執行相應次數的 {CreateRoom} 的中繼功能函式，因此與其說分別執行了一次 {CreateRoom} 與 {CreateSubRoom}，我想以功能函式 {CreateRoom} 作為生成程序結果中主要與唯一的說明主體會比較清楚。

在圖 4-1 中的 $n = 2$ 和 $n = 3$ 兩個範例，系統程式分別執行了 12 次與 24 次 {CreateRoom}，其中關於這些數字，必須注意它們與系統從 roomPool 取出的資料以及每一層生成疊代產生的房間樣式有必然關係。每間房間的樣式除作為 axiom 的起始房間由使用者設定，其餘的房間皆交由系統參照邊界和區域挖空等

使用者設定，以及比對鄰近相關的未生成與已生成區域、加以考察周遭環境後，產生半隨機的樣式選擇結果，因此無法說 $n = 2$ 和 $n = 3$ 的疊代狀況下會必然引用與執行 12 次與 24 次{CreateRoom}，圖中範例僅表現出一種理想的可能結果。

因為每次往周遭生成新的房間是以隨機的方式取得樣式，且棋盤大小與初始位置可由使用者定義，使得關卡地圖較常出現如圖 4-2 等不規則外型的結果樣貌，另外透過使用者設定也可產生類似橫向卷軸的地圖，如圖 4-2 右側的關卡生成結果。圖中以黃色邊框標記的部分代表初始房間，在圖 4-2 更以藍色符號標出玩家起始位置；在系統預設上玩家一開始會從這個位置出發，這是因為考量了關卡中所有的房間裡只有初始房間會絕對存在。

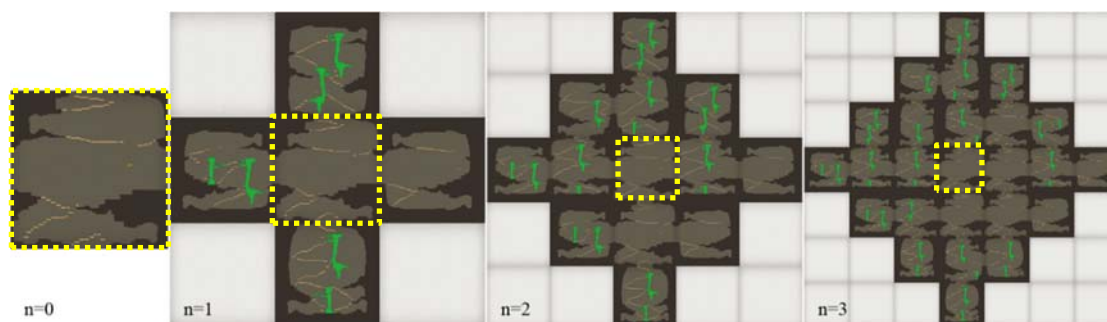


圖 4-1 基本疊代的關卡生成範例

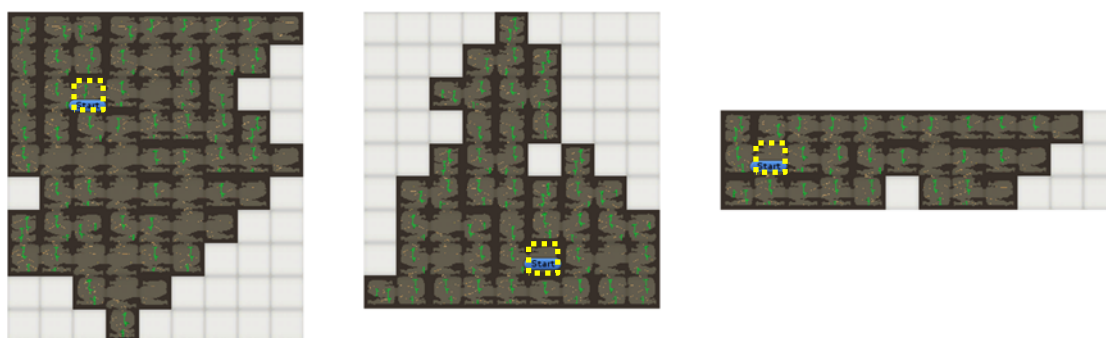


圖 4-2 不規則外型的關卡生成範例

在圖 4-3 中可以看到以隨機取樣的方式，將最後一輪生成的房間選擇數個做為終點所在的房間，可以看到較完整的遊戲路線，黃色路線便是一條玩家可能經過的路徑。設定如此大量的出口主要用途是提供大量的路徑解答、避免玩家在過

大的地圖中盲目探索過久，導致遊玩興致下降，當然也可僅製作一個出口，例如只選擇黃色的出口保留，這中間的取捨由使用者決定。

圖 4-4 中呈現出兩個同樣使用者設定產生的不同結果，兩張地圖使用相同的起始點、同樣 5×20 大小的棋盤，並一樣限制遞迴次數為 25 以及挖空一模一樣的棋盤區塊，這其中的差異來自於房間的樣式選擇上採完全隨機，因此有可能會出現過早封閉房間路徑，導致房間佔地面積比預期的小，尤其偏地圖後段且與前段區域有只一間房間做連接的地方，容易發生因選擇的樣式開口位置導致中斷房間路徑延續的狀況，而如 4-4 上圖中完全依照有挖空與沒挖空的區域產生房間路徑，在隨機生成過程中實際出現機率的非常低。



圖 4-3 多重出口的關卡生成範例



圖 4-4 相同自定義參數產生的不同關卡生成比較範例

第三節 參數變化

本研究範例中定義每間房間尺寸為 40.96×40.96 、最小間距單位為 0.32，並以房間正中央中心點作為內部物件對齊的座標系，而在內部物件預設上，平台有 1.28×0.32 與 0.64×0.32 兩種規格，敵人與玩家角色則同樣定為 1.28×1.28 ，且於角色物件下方如圖 4-5 所示的位置綁上寬度遠小於最小間距單位的碰撞偵測體 (Collider)，其他的陷阱與光球物件大小分別為 0.32×0.16 與 0.64×0.64 。

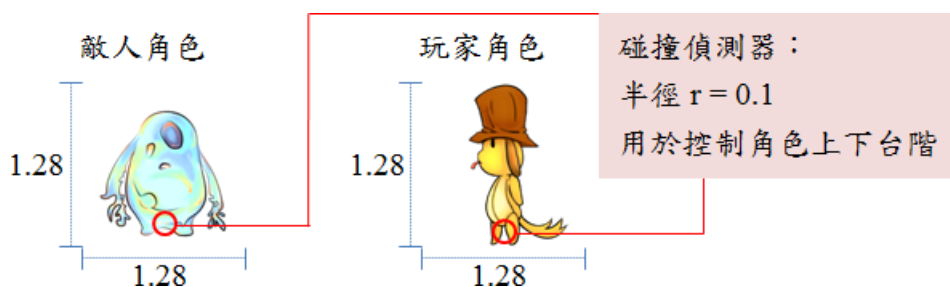


圖 4-5 範例遊戲中角色示意圖

在這一小節中將分別介紹房間模板上各個參數表對房間內物件布局的影響，並藉由實際的生成範例說明房間內部物件的生成過程與難易度參數調整產生的變化。以下將從產生房間內背景 surfaces 開始說明。

一開始產生房間時，系統會根據選定的房間樣式選擇其樣式包含的樣板，從這些樣板中決定了合適的模板才開始真正生成遊戲中房間的實體，而 surfaces 與背景物件便是房間的基本視覺外觀。surfaces 做為房間底層的平台集合為靜態剛體，與背景物件一樣不隨難易度變化，在同一個模板中包含一個固定的底層通道、0 到 2 個通用牆柱元件、0 到數個移動台階、零星定點的小平台與一張背景貼圖，可參考如圖 4-6 中呈現出屬於樣式編碼為 8 下固定樣板的視覺外觀。

為了簡化作業流程與減少地形測試，研究範例中以 0.32×0.32 為單位構築底層背景 surfaces，以降低玩家在遊玩中可能卡在畫面的狀況，這使得範例遊戲畫面像是以圖磚拼貼而成，但實際上背景貼圖與剛體可以個別拆開，使用者在未來可將 surfaces 的貼圖更換成較有設計美感的或以多種小型物件集合群組取代目前畫面上的視覺效果，當然也可以維持以圖磚的方式構圖，並額外增加以程序排列圖磚繪製房間畫面的功能。

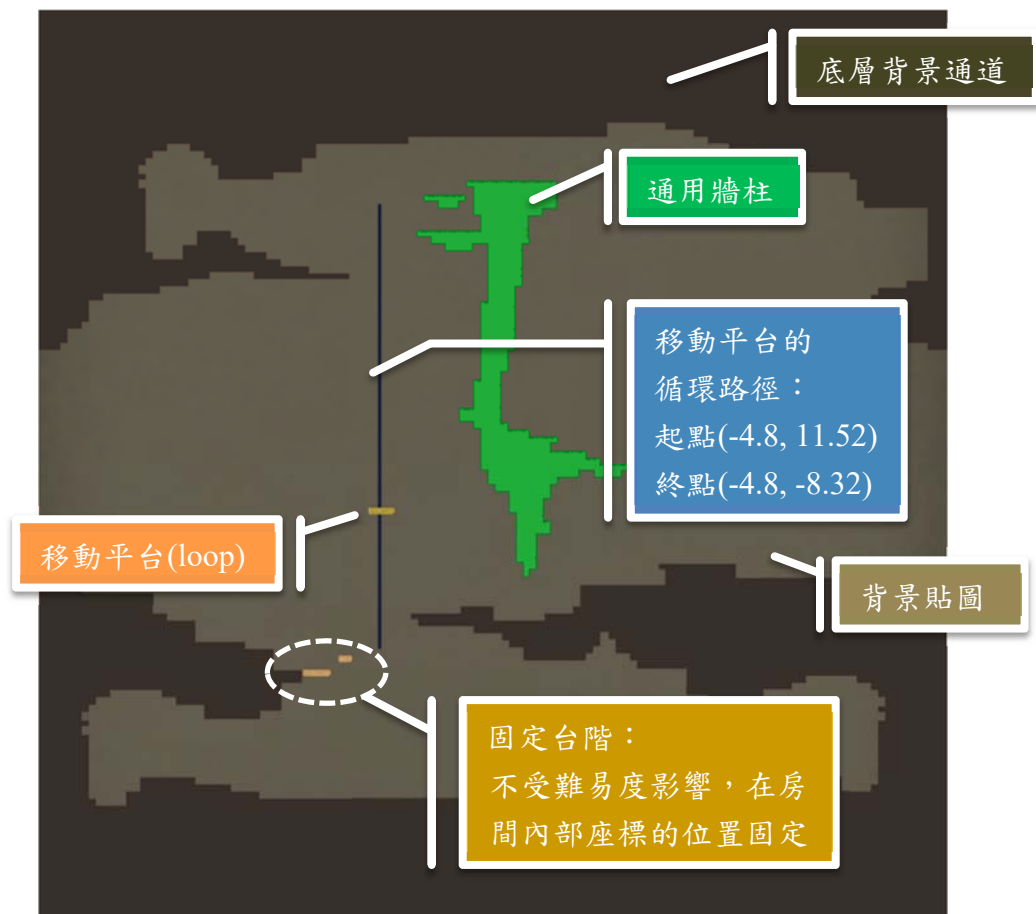


圖 4-6 房間內部背景與固定物件的生成範例

以下將以圖 4-6 的房間為例，依序說明實作上台階路線、敵人預定生成區與陷阱預定生成範圍該如何規劃。為避免混淆，除了用作物件座標基準參考的背景底圖，在這些生成規劃的範例中只會單獨呈現用於說明的元件，例如在台階生成的範例圖上不會看到敵人物件，雖然實質程序上這些物件在生成時是一同透過樣板呼叫產生的。

另外，使用者可以自己的需求關閉部分物件的生成程序，亦即將指定的元件從房間樣板中刪除，但並不建議這麼做，尤其是嚴重影響玩家能否從起點走到終點的平台路徑，有可能造成遊戲喪失可玩性與遊戲性。

針對房間內部物件的生成，首先關於台階路線設定可參考圖 4-7，可以看到兩條依照各自的起終點連線的紅色線段，這些線段的起終點座標值可參考表 4-2，而如表 4-2 這樣形式的參數表將被統整至房間樣板；在房間生成時便能讀取該表單並藉由難易度控制產生台階路徑上應有的平台。

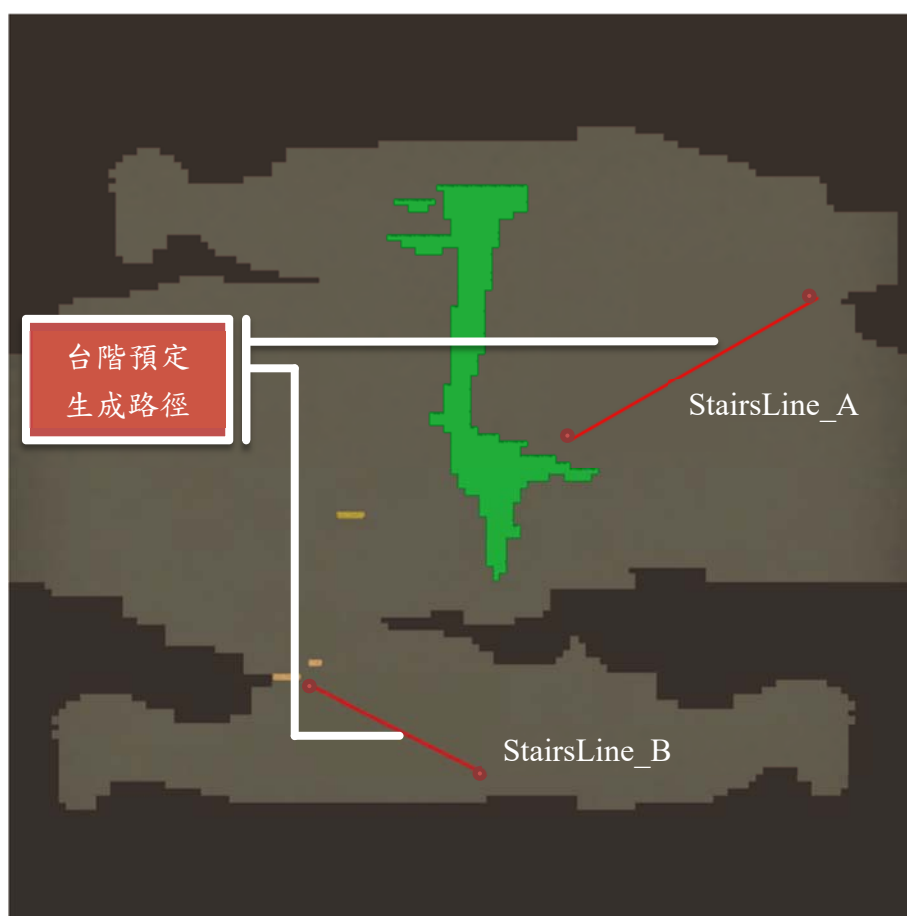


圖 4-7 房間內部台階生成路線規劃範例

表 4-2 生成台階的路線規範表

台階路線代號	起始端點	結束端點
StairsLine_A	(5.12, 1.28)	(16.32, 7.68)
StairsLine_B	(-6.4, -9.92)	(1.28, -13.76)

房間內台階路徑受難易度影響的具體例子可以參考圖 4-8~4-12，以下這些圖將依序展示根據 0~4 的難易度參數產生的台階間距變化。



圖 4-8 難易度 0 的台階生成範例



圖 4-9 難易度 1 的台階生成範例

在範例中兩種不同尺寸的平台為隨機出現、機率比為 1：1。在圖 4-8 與圖 4-9 的平台差異看起來似乎很小，但隨著難易度上升差異便會慢慢明顯。

另外在圖 4-9 左上產生了一個較高的台階，這是因為當路徑生成快接近終點時，系統會檢查路徑上倒數第二階與目標終點的差值，如果距離過大、可能依目前的間距大小無法確實在最後一階連接到終點，並且角色的跳躍能力確實無法彌補的情況下，系統會將最後一階的位置更動，而如果以間距最大值位移後仍沒改善，系統會確認目前的間距過大而開始產生額外的台階。

在圖 4-9 中可以看到右下的路徑最後一階相對高出許多，因為生長的台階已抵達路徑上作為結束的端點 X 值，但此時最後的台階與原本期待的終點位置間距並非大於玩家可跳過的高度，而且也接近地面、不宜產生額外的台階，如果強制讓末端台階對齊終點座標，反而會導致與上階台階的垂直間距過高。



圖 4-10 難易度 2 的台階生成範例



圖 4-11 難易度 3 的台階生成範例

通常額外產生階梯會發生在垂直間距過大的處理上，包含因為難易度變化帶來的影響，可參考難易度較高的圖，例如圖 4-11。水平間距的末端狀況處理相對較少，只有像是間距小於大台階的寬度時會 100%選擇小台階，以及視情況捨棄依難易度參數挑選間距而由系統決定最後一階的位置，例如當距離終點小於 1.6，那麼就算是在難易度 4 的規則生成下也不會發生選用加上台階寬度必然會出過範圍的 1.28 間距。這些末段台階處理與生成範圍限制非常重要，假使放任台階生長至限定路徑外可能會發生意外狀況，例如可能波及相鄰的台階路徑產生不合理的交疊錯位，嚴重的狀況可能會導致因為路徑損壞而無法進行遊戲。



圖 4-12 難易度 4 的台階生成範例

接著說明敵人物件的設定方式，它跟平台物件一樣都有一個存放參考座標的參數表，這個表也同樣會被合併整理至房間模板中作為日後所有同樣式模板下的敵人生成依據，該表定義如表 4-3，針對一般不會飛、在地面上行走的敵人怪物

將生成範圍定義成以起終點標記的線段，假使是用作產生會飛或浮空的對象，則改用預定範圍的中心點與邊界的長寬做限制。

敵人生長預定地範圍與生成台階的路徑規範上最主要的差異在於所有的參考區域沒有百分之百的使用要求；依據房間背景中剛體的擺設，有些路徑勢必得絕對存在，但敵人的座標設置就完全取決於使用者的喜好，可僅設置一個大範圍的區塊做為生成敵人物件的座標取樣空間，或者分散成數個小型區域讓實際生成敵人物件時不會發生太過壅擠的狀況，甚至可以用參數控制每個區域被選中的機率。系統在初始設定上所有房間內的敵人預定生成區的選用機率均等。

以下範例將以敵人只能同玩家做簡單的跑跳定義房間內敵人預定生成的區域，因為敵人本身具有重量、不會浮空，在生成後就算座標位置不在地板上也會安全落定，在圖 4-13 中以灰紅色的區塊標示紅色線段下敵人產生後可能站立的位置。敵人受難易度影響的具體例子將整理在下圖 4-14~4-18，可前後對照圖中的敵人依隨機亂數各自產生在表 4-3 中預定的生成區塊。

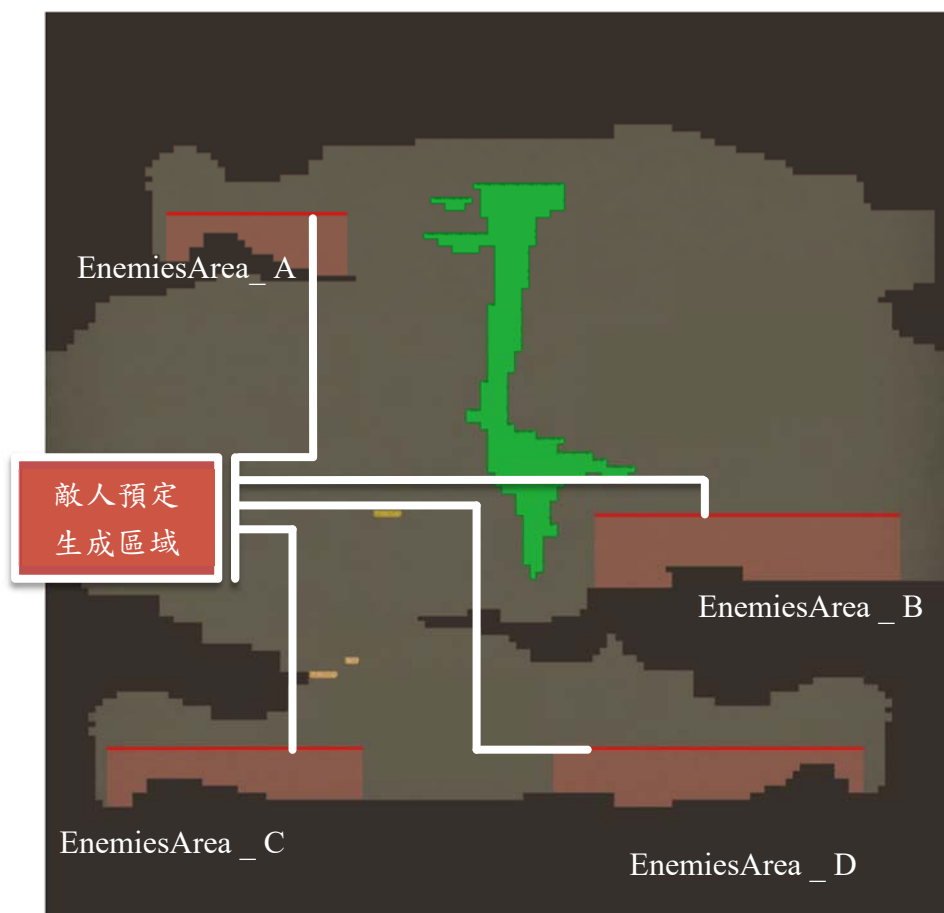


圖 4- 13 房間內部敵人預定生成區域規劃範例

表 4- 3 敵人預定生成區域規範表

敵人區域代號	起始端點	結束端點
EnemiesArea_ A	(-15.04, 11.2)	(-6.88, 11.2)
EnemiesArea_ B	(4.48, -2.08)	(18.08, -2.08)
EnemiesArea_ C	(2.72,-12.48)	(16.32,-12.48)
EnemiesArea_ D	(-17.92,-12.48)	(-6.24,-12.48)



圖 4- 14 難易度 0 的敵人生成範例



圖 4- 15 難易度 1 的敵人生成範例



圖 4- 16 難易度 2 的敵人生成範例



圖 4- 17 難易度 3 的敵人生成範例



圖 4-18 難易度 4 的敵人生成範例

陷阱的生成參數設定上與敵人大致雷同，只是建議上不要將可作為生長區域的預定線段設置過長，因為可能會發生陷阱過度集中或並排在一起的情況。假如是發生在擁有動態的敵人物件生成上，因為敵人本身可自由活動，因此就算原本初始位置很集中，但隨著時間會漸漸分散。陷阱位置設置不恰當的話可能還會一下將遊戲難度拉得太高，除非使用者是刻意想製造這樣的效果，例如無縫隙地並排一整排的陷阱在台階下方，作為懲罰玩家跳躍失誤的下場。

下方陷阱生成範例便以陷阱只能生長於地面上、不會浮空作為預定生長區域的設定基準，因範例中選用的預定線段較多且狹窄，將以白色虛框加以標示，如圖 4-19，在後續陷阱生成範例上也會同樣標記白框以利辨識。而陷阱受難易度影響的例子將依照難易度 0~4 的順序整理在圖 4-20~4-24，可前後對照圖中的陷阱依隨機亂數各自產生在表 4-4 中預定的生成區域線段上。

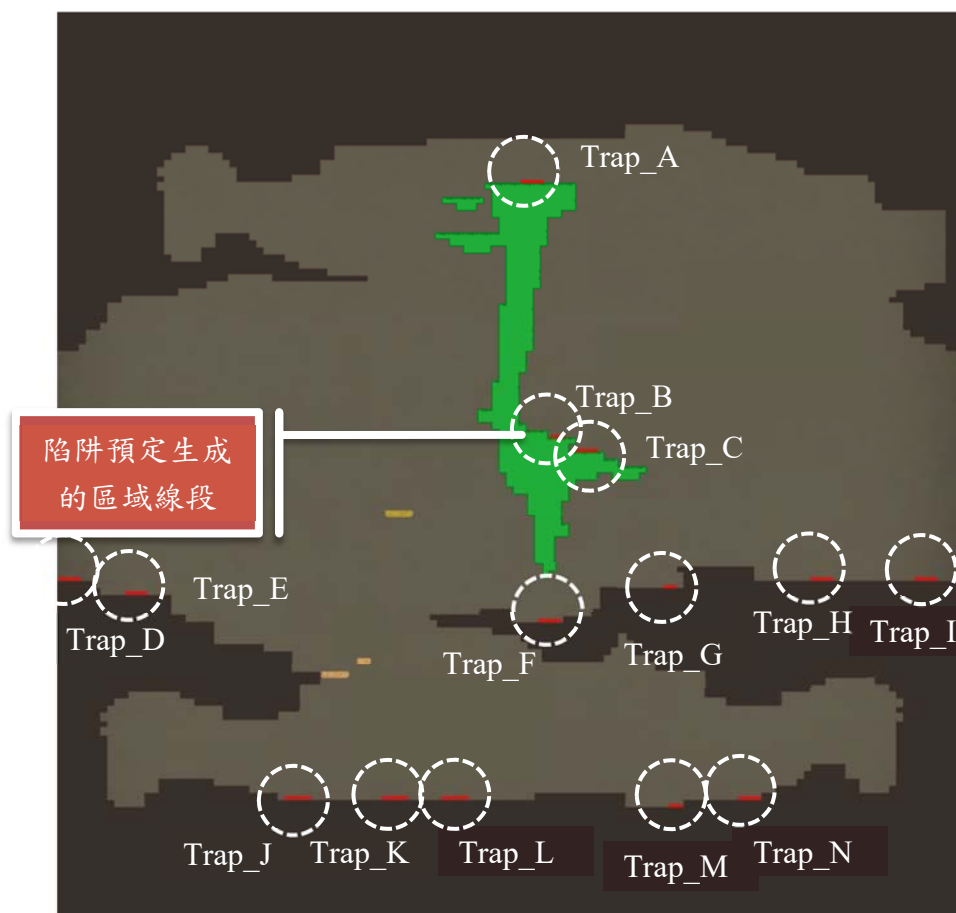


圖 4- 19 房間內部陷阱預定生成區域規劃範例

表 4-4 陷阱預定生成區域規範表

陷阱區域代號	起始端點	結束端點
Trap_A	(0.48, 12.8)	(1.44, 12.8)
Trap_B	(1.92, 1.28)	(2.88, 1.28)
Trap_C	(3.52, 0.64)	(4.48, 0.64)
Trap_D	(-20.48, -5.12)	(-19.52, -5.12)
Trap_E	(-17.6, -5.76)	(-16.64, -5.76)
Trap_F	(0.96, -7.04)	(1.92, -7.04)
Trap_G	(5.44, -5.44)	(6.08, -5.44)
Trap_H	(13.6, 5.12)	(14.56, 5.12)
Trap_I	(18.24, 5.12)	(19.2, 5.12)
Trap_J	(-10.24, -15.36)	(-8.96, -15.36)
Trap_K	(-5.76, -15.36)	(-4.48, -15.36)
Trap_L	(3.04, -15.36)	(1.76, -15.36)
Trap_M	(7.04, -15.68)	(7.68, -15.68)
Trap_N	(10.24, -15.36)	(11.2, -15.36)



圖 4-20 難易度 0 的陷阱生成範例



圖 4-21 難易度 1 的陷阱生成範例



圖 4-22 難易度 2 的陷阱生成範例



圖 4-23 難易度 3 的陷阱生成範例

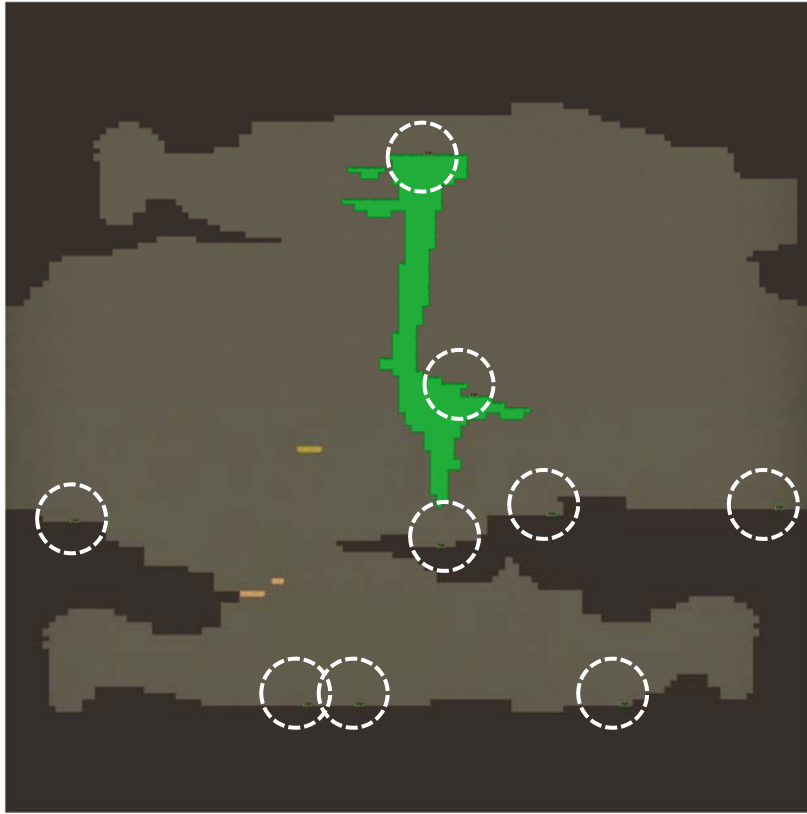


圖 4- 24 難易度 4 的陷阱生成範例

在設計上作為收集品的光球並沒有需要像平台與障礙元件要特別規範，由使用者自行決定該放置的位置，或者綁在台階與敵人的生成程序中，讓玩家在收集上必須面對遊戲中的障礙挑戰，因此不額外制定不同座標群的參數表。在圖 4-25 中將同時展示以難易度 0 生成的台階路線，以難易度 3 產生的敵人和用難易度 4 設置的陷阱，加上一組以固定位置設置的光球集合。

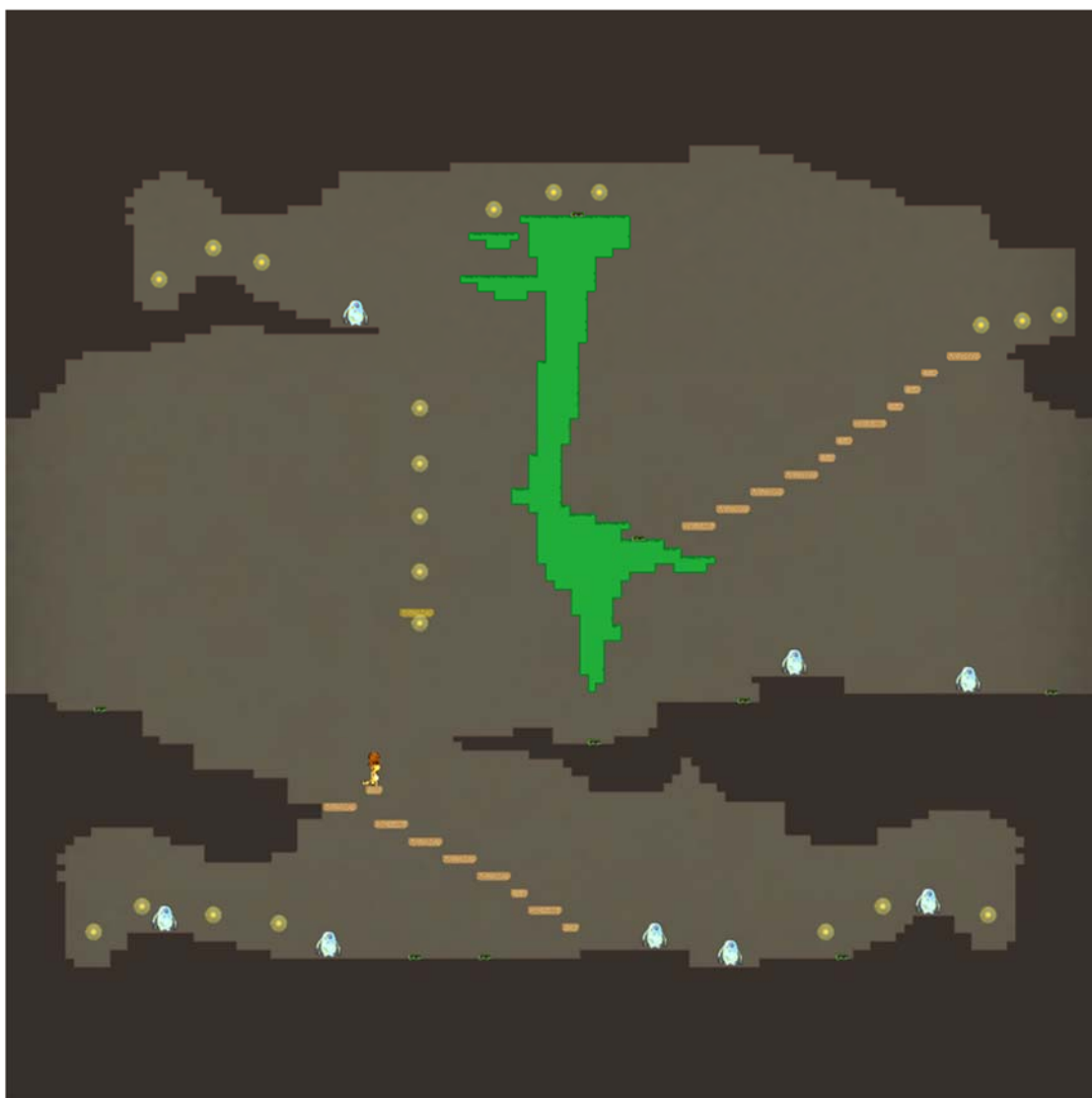


圖 4-25 房間內部物件的生成範例

第五章 結論與建議

本章將就實作上，針對本研究以 2D 平台遊戲設計模型為基礎套用 L-system 下程序性生成的關卡進行結果討論。並由上一章所做的實驗結果來進行更進一步的分析與建議。

第一節 結論

本研究主旨在設計一套以 L-system 為基礎的 2D 平台遊戲關卡自動生成機制，以求減少遊戲開發時間與資源成本。在整個生成程序上，有幾項設計重點，分別是邊界容錯處理、難易度處理以及使用 2D 平台遊戲關卡設計模型做為 L-system 生成規則的定訂基礎，包括以此衍伸的房間樣板規範。本節中將詳細的以這些項目進行討論。

一、關卡生成規則

本研究以現有的 2D 平台關卡架構模型作為劃分遊戲組成元件與訂定生成規則的基礎。從第四章中可以看到以此關卡生成規則產生的關卡結果，就解決路徑問題、建構大型遊戲場域以及組織房間內部的元件都有良好表現。這一方面因為 L-system 遞迴生成的特性，以及遊戲關卡內元件間彼此的關係邏輯明確，也因為做為參考的關卡概念模型與架構組織表明關卡設計本身便是由多種元件層層堆疊產生，因此格外適合用 L-system 構築出整體關卡生成的程序性架構，這其中也反應出 L-system 中規則訂定的重要性。

本研究以夾帶許多關卡元件的空間區塊，也就是房間，作為生成對象，因為房間本身便具備空間連貫的性質，因此只要生成規則邏輯正確，不怕周遭沒有可以連通的道路，就算拿來構築大型迷宮地圖也沒有問題。另外，以本研究的架構生成關卡時也不用為了確保關卡內部起終點的路徑問題額外計算解決路徑，因為在該架構下生成的關卡地圖並不存在封閉區域；房間無法連接的地方也不會有其他房間存在。

二、邊界容錯處理

在棋盤空間管理上，本研究透過使用者設定與一連串容錯處理，大幅將低棋盤占用的空間資源，另一方面這也提高使用者設計棋盤外觀的彈性與自由度；使用者可藉由本研究提供的邊界、初始位置與挖空區域等設定，進一步創造出具個人特色的關卡地圖，像是第四章中展示的範例地圖。

三、難易度處理

在遊戲設計上難易度控制是很重要的角一環，遊戲設計的太簡單玩家會覺得無聊，太難則也不好玩。本研究將控制遊戲難度的幾項元件分別訂定在房間模板上，包含平台與障礙。使用者可透過調整這些從背景元件中獨立出的物件，並搭配預設參數表進一步系統化地控制關卡遊戲中的難度。

藉由個別操作房間的難易度值，可以在同一遊戲關卡中設計在出有循序漸進的挑戰、幫助玩家融入遊戲世界。參數表中保留很大的空間供使用者發揮，在第四章中可以看到藉由調控參數生成的範例房間。使用者也可以透過組織不同物件的難易度變化增加新的遊戲關卡，有效利用不同難易度下的不同物件組合排列出豐富多樣的可能挑戰。

四、房間樣板

房間模板整合了部分難易度設定以及房間內部保有的物件群，每間房間以所屬的房間模板確保各自房間中必要的功能元件與關卡道具。在這邊便是人為控制最主要展現的地方；從生成物件、物件外型、貼圖、難易度調整、座標位置、生成機率等等，都是可自由變換的參數，如果有新的關卡元件想加入遊戲中，也是定義在這個模板裡。房間模板可說是為了提供使用者自定義而產生的，在一些討論文獻中也同樣說遊戲中模板是遊戲設計師最重要發揮的地方。

透過房間模板來創建關卡房間內物件，可以確保有系統的組織這些元件，並保有人性設計的特色。使用者在設計房間模板時不妨可站在玩家的角度思考。

從上述的機制設計中，可以看到本研究提出的關卡生成程序提供使用者控制包含邊界設定、難易度設定與房間樣板設計等主要三個項目，在更細節的部分還有出入口位置與數量設定、生成遞迴次數宣告與房間內部物件的自定義(?)等等。

在主要項目下的一些分支或共用的設計元件，如房間內部物件，建議設計師根據實際需求，調整該房間內部物件在難易度設定與房間樣板設計上相對的參數關係、取得適當的平衡，也因為本研究制定的生成辦法將人為可控制的部分以參數表各自分開，並在程序引用上盡量採取互不干涉的做法，因此使用者在使用上需多加留意；系統無法偵測出所有人為設計上的矛盾，尤其是牽涉到美感與趣味性質的感性設計。

雖然加入人為控制的部分能大大提高本研究的生成系統於不同遊戲上的適應性，並增加關卡生成內容的豐富度與人性化，但同時使用者在制定生成相關的參數與設計元件與模組時，會比完全採用程序化的設計方式來的更加費神，如何在人為與系統控制上找出適當的平衡點是未來很重要的課題。在現階段，使用者在運用本研究提供的生成系統時，不妨參考預設的模組與參數表，取自己所需的部分進行微調或修改，一步步拿捏出自己需要控制的比例。

第二節 後續建議與未來工作

依據文獻整理的關卡設計架構，2D 平台遊戲的關卡構成元素不單只有本研究程序生成上所使用的平台、障礙、收集品等物件，本研究也僅是為用 L-system 生成關卡提供一條可行的參考路線，設計師可依未來實際需求增減所有的關卡元件，或者套用至其他設計模型與遊戲框架中，思考各種遊戲元素在構成遊戲上的邏輯關係。

本研究就本質上來說，較偏向以 Smith 模型中的挑戰群(Group)作為生成對象，雖然房間內的階梯、敵人、陷阱等元件也包含在生成程序中，但主要還是對直接構成關卡的房間群作生成規劃。未來可考慮將 L-system 的概念延伸到遊戲設

計上下層面，以不同的關卡元素或物件群為生成對象，整合關卡架構與內部元件創建至生成規則中，以完全自動化生成遊戲關卡為目標。

另外，目前的房間樣板仍顯粗糙，樣板種類也非常少，大大降低程序生成上多樣變化的優點。做為改良，關卡生成上的參數表可以再進一步優化以貼近實際需求，一些參數的設定往往不夠直覺，必須反覆確認實際生成的效果，在未來工作中希望能做出方便設計師操作的 UI 介面，以利於加速整體設計流程並大量產出可用的樣板。

參考文獻

- Adams, T. & Adams, Z. (2006). *Slaves to Armok II: Dwarf Fortress* (PC Game). *Bay 12 Games*.
- Adelhardt, K. & Kargov, N. (2012). *Mario Game Solver*. *KindSoftware: Software Engineering with Applied Formal Methods*.
- Angband Development Team. (1990). *Angband* (PC Game).
- Apperley, T. H. (2006). Genre and game studies: Toward a critical approach to video game genres. *Simulation and Gaming* 37(1).
- Benoit-Koch, F. (2014). *Jack Benoit* (App).
- Benoit-Koch, F. (2014). Procedural Level Generation for a 2D Platformer. Retrieved from <http://fbksoft.com/procedural-level-generation-for-a-2d-platformer/>
- Berens, K., & Howard, G. (2001). *The Rough Guide to Videogaming 2002*. *London and New York: Rough Guides*.
- Bethesda. (1996). *The Elder Scrolls II: Daggerfall*(PC Game).
- Bizzard North. (1996). *Diablo II* (PC Game). *Bizzard Entertainment*.
- Bjork, S., & Holopainen, J. 2004. Patterns in Game Design. *Charles River Media, ch. 2*, 7–32.
- Blow, J. (2008). *Braid* (XBLA). *Number None*.
- Boutros, D. (2006). A Detailed Cross-Examination of Yesterday and Today's Best-Selling Platform Games. *Gamasutra* (August). Retrieved from http://www.gamasutra.com/features/20060804/boutros_01.shtml.
- Braben, D. & Bell, I. (1984). *Elite*(Acorn). *Acornsoft*.
- Byrne, E. (2005). *Game Level Design*. *Charles River Media*.

- Compton, K. & Mateas, M. (2006). *Procedural Level Design for Platform Games*. In Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '06).
- Crawford, C. (1984). *The Art of Computer Game Design*. Berkeley: Osborne/McGraw-Hill.
- Duarte, P, M., & Rodrigues, S. (2014). Goal-Oriented Procedural Level Generation for Action/Adventure Games. Retrieved from <http://www.imagina.dimap.ufrn.br/imagina/projeto/12/goal-oriented-procedural-level-generation-for-actionadventure-games>
- EA-Maxis. (2008). *Spore* (PC Game). *Electronic Arts*.
- Farbrausch. (2004). *.kkrieger*(PC Game).
- Firaxis Games. (2005). *Sid Meier's Civilization IV* (PC Game).
- Firaxis Games. (2012). *XCOM: Enemy Unknown*(PC Game). *2K Games*.
- Fisher, J. (2015). How to Make Insane, Procedural Platformer Levels. Retrieved from http://www.gamasutra.com/view/feature/170049/how_to_make_insane_procedural_.php?print=1
- Flagship Studios. (2007). *Hellgate: London* (PC Game). *Electronic Arts*.
- Gearbox. (2012). *Borderlands2* (PC Game). *2K Game*.
- Ghignola, A. (2000). *Noctis* (PC Game).
- Hullett, K. & Mateas, M. (2009). *Scenario Generation for Emergency Rescue Training Games*. In Proceedings of the 4th International Conference on Foundations of Digital Games. Orlando, FL. April 26 – 30, 2009.
- I-Novae Studios. (2010). *Infinity: The Quest for Earth* (MMOG).

- Jennings-Teats, M., Smith, G., & Wardrip-Fruin, N.(2010). *Polymorph: dynamic difficulty adjustment through level generation*. PCGames '10 Proceedings of the 2010 Workshop on Procedural Content Generation in Games.
- KCET. (1997). Castlevania: Symphony of the Night (PS). *Konami*.
- King, G., & Krzywinska, T. (2002). ScreenPlay: Cinema/videogames/interfaces. *London: Wallflower Press*.
- Konami. (1987). Contra (Amstrad CPC).
- Koncewicz, R. (2011). Super Mario Bros 3 Level Design Lessons. Retrieved from <http://www.significant-bits.com/super-mario-bros-3-level-design-lessons>
- Larsen, S. (2006). Level Design Patterns. *Information Science in April 2006 at the IT-University of Copenhagen*.
- Manousakis, S. (2006). Musical L-systems. *Master's Thesis-Sonology, The Royal Conservatory, The Hague, 2006*.
- Martin, G. A., Schatz, S., Hughes, C. E. and Nicholson, D. (2010). *The use of functional L-systems for scenario generation in serious games*. PCGames '10 Proceedings of the 2010 Workshop on Procedural Content Generation in Games.
- Marvie, J., Perret, J. and Bouatouch, K. (2005). The FLsystem: A Functional L-system for procedural geometric modeling. *The Visual Computer*. 21, 5, 329–339.
- Měch, R., Prusinkiewicz, P. (1996). *L-systems: from the Theory to Visual Models of Plants*. SIGGRAPH '96 Proceedings of the 23rd annual conference on Computer graphics and interactive techniques.
- Milam, D., & Seif El-Nasr, M. (2010). *Analysis of Level Design 'Push & Pull' within 21 games*. FDG '10 Proceedings of the 5th International Conference on the Foundations of Digital Games.

- Milam, D., & Seif El-Nasr, M. (2010). *Design Patterns to Guide Player Movement in 3D Games*. Sandbox '10 Proceedings of the 5th ACM SIGGRAPH Symposium on Video Games.
- Moby Games Genre definitions. <http://www.mobygames.com/glossary/genres>
- Müller, P., Wonka, P., Haegler, S., Ulmer, A. and Van Gool, L. (2006). *Procedural modeling of buildings*. ACM Transaction on Graphics (SIGGRAPH Proceedings). 25, 3, 614-623.
- Nelson, M. & Mateas, M. (2007). *Towards Automated Game Design*. Proc. 10th Congress of the Italian Association for Artificial Intelligence (AIIA 2007), Rome, Italy. September 10-13, 2007.
- Nelson, M. (2007). Breaking Down Breakout: System and Level Design for Breakout-style Games. Gamasutra (August). Retrieved from http://www.gamasutra.com/view/feature/1630/breaking_down_breakout_system_and_.php.
- Nintendo. (1985). Super Mario Bros (NES).
- Nintendo. (1996). Super Mario 64 (N64).
- Novak, J. (2006). Game Development Essentials. *Thomson Delmar Learning*.
- Oddlabs. (2005). Tribal Trouble(PC Game). *GarageGames*.
- Parish, Y, I, H. & Müller, P. (2001). *Procedural modeling of cities*. SIGGRAPH '01 Proceedings of the 28th annual conference on Computer graphics and interactive techniques.
- Prusinkiewicz, P. & Lindenmayer, A. (1990). The algorithmic beauty of plants. *Springer-Verlag*.
- Pwnee Studios. (2013). Cloudberry Kingdom (PS). *Ubisoft*.
- Rare. (1994). Donkey Kong (GB). *Nintendo*.

- Rare. (1999). Donkey Kong64 (N64). *Nintendo*.
- Rovio Entertainment. (2009). *Angry Birds (App)*.
- Salen, K., & Zimmerman, E. (2004). Rules of Play: Game Design Fundamentals. *MIT Press, Cambridge, Massachusetts, ch. 23, 313–327*.
- Sega. (1991). Sonic the Hedgehog (SG).
- Smith, A., R. (1984). *Plants, fractals, and formal languages*. SIGGRAPH '84 Proceedings of the 11th annual conference on Computer graphics and interactive techniques.
- Smith, G., Cha, M., & Whitehead, J. (2008). *A Framework for Analysis of 2D Platformer Levels*. Sandbox '08 Proceedings of the 2008 ACM SIGGRAPH Symposium on Video games.
- Smith, G., Gan, E., Othenin-Girard, A., Whitehead, J. (2011). *PCG-Based Game Design: Enabling New Play Experiences through Procedural Content Generation*. PCGames '11 Proceedings of the 2nd International Workshop on Procedural Content Generation in Games.
- Smith, G., Treanor, M., Whitehead, J., & Mateas, M. (2009). *Rhythm-Based Level Generation for 2D Platformers*. FDG '09 Proceedings of the 4th International Conference on Foundations of Digital Games.
- Smith, G., Whitehead, J., & Mateas, M. (2010). *Tanagra: A Mixed-Initiative Level Design Tool*. FDG '10 Proceedings of the 5th International Conference on the Foundations of Digital Games.
- The NetHack Dev Team. (1987). *NetHack (PC Game)*.
- Thompson, J. (2011). Procedural Generation-The Caves. Retrieved from <http://noelberry.ca/2011/04/procedural-generation-the-caves/>
- Toy, M., Wichman, G., and Arnold, K. (1980). Rogue (PC Game).

- Uber Entertainment. (2014). Planetary Annihilation(PC Game).
- Udisoft. (2008). Far Cry 2(PC Game).
- Valve Corporation. (2008). Left 4 Dead(PC Game).
- Wolf, M. J. P. (2001). The Medium of Video Game. *Austin: University of Texas Press.*
- Yu, D., Hull, A. (2013). Spelunky (PC Game). *Mossmouth.*
- Zagal, J. P., Mateas, M., Fernandez-Vara, C., Hochhalter, B., & Lichti, N. (2005). *Towards an Ontological Language for Game Analysis.* In Proceedings of the Digital Interactive Games Research Association Conferences (DiGRA 2005).