



國立台灣科技大學

資訊工程系

碩士學位論文

以遊玩特徵為導向的程序化內容生成方法

Game Design Goal Oriented Approach for Procedural
Content Generation

研究 生：王澤浩

學 號：M10415096

指導教授：戴文凱博士

中華民國一百零六年七月二十七日

中文摘要

中文摘要，編輯中。



ABSTRACT

Ab. Under construction.



誌謝

誌謝，頁面保留。



目 錄

論文摘要	I
Abstract	II
誌謝	III
目錄	IV
圖目錄	VI
表目錄	VIII
演算法目錄	IX
1 緒論	1
1.1 研究背景與動機	1
1.1.1 迷宮探索遊戲 (dungeon crawl) 類型介紹	1
1.2 研究目的與研究問題	1
1.2.1 研究目的	2
1.2.2 研究問題	2
1.2.3 預計研究貢獻	2
1.3 本論文之章節結構	2
2 相關研究	3
2.1 含有敘事脈絡的關卡	3
2.2 關卡生成的方法	3
2.2.1 程序化生成任務內容	3
2.2.2 程序化遊戲物件擺放	5
2.3 使用關卡生成技術的遊戲	6
2.3.1 遊戲 - Spelunky	7
2.3.2 遊戲 - The Binding of Isaac	7
2.4 基因演算法	7
3 研究方法	8
3.1 任務語法	8
3.1.1 建立任務符號表	9
3.1.2 建立任務規則	10
3.1.3 產生任務圖	13
3.2 空間建構	16

3.2.1	基礎結構	16
3.2.2	端點識別物	17
3.2.3	房間容器類型	18
3.2.4	從任務圖轉為遊戲空間	20
3.3	地圖片段	23
3.3.1	基因的儲存結構	23
3.3.2	演化之適應性函數	24
3.3.3	制衡取向的適應性函數	28
3.3.4	負數權重的適應性函數	28
3.3.5	多項適應性函數合併	29
3.3.6	套用物件演化機制的房間容器	30
4	實驗結果與分析	33
4.1	實驗定義	33
4.1.1	遊戲操作說明	33
4.1.2	遊戲物件說明	33
4.1.3	實驗參數設定	34
4.2	資料收集	34
4.3	演化結果與其品質	34
4.4	標準化結果之比較	35
4.5	房型規模之比較	36
5	結論與後續工作	37
5.1	貢獻與結論	37
5.2	限制與後續工作	37
	參考文獻	38
	附錄一：The binding of Isaac 的任務語法	39
	附錄二：鎮守要道情境之演化釋例	40
	附錄三：狹路驅逐情境之演化釋例	41
	授權書	42

圖 目 錄

圖 2.1 心流體驗應用於遊戲設計	3
圖 2.2 Mission/Space 框架之主要結構	4
圖 2.3 Unexplored 的關卡設計	5
圖 2.4 Antonios Liaps 提出的兩階段式關卡演化	6
圖 3.1 本論文提出系統之流程圖	8
圖 3.2 Dungeon Generator 工具	9
圖 3.3 利用遊玩特徵建立任務符號表之範例	10
圖 3.4 線性任務規則範例，建立主線任務	11
圖 3.5 建立主線任務的預期任務圖與空間預覽	11
圖 3.6 非線性任務規則範例，設置特殊房	12
圖 3.7 非線性任務規則範例，建立支線任務	12
圖 3.8 非線性任務規則範例，鎖的提前出現與房間攤平	12
圖 3.9 於 Dungeon Generator 工具中，若規則為非法狀態將不予以生效	13
圖 3.10 最終輸出的任務圖	15
圖 3.11 影響遊戲性的遊戲物件，由左至右分別為敵方、寶箱與陷阱	16
圖 3.12 房間容器的建構情形，可觀察其由體素單位所構築而成	17
圖 3.13 房間容器時所使用的五種裝飾物	17
圖 3.14 端點識別物的實際使用與串接情形	18
圖 3.15 房間類型 - 由左至右為入口、出口與魔王房	18
圖 3.16 房間類型 - 由左至右為 I 型、L 型、T 型與 + 型通道	19
圖 3.17 房間類型 - 由左至右為石牆、秘密房、鎖、商店與寶藏房	19
圖 3.18 房間類型 - 死路	19
圖 3.19 建造方式，任務終端節點與房間容器對應之範例	21
圖 3.20 已疊代生成的任務圖，不同亂數種子轉換為對應結果的遊戲空間	22
圖 3.21 經過剩餘空間替換後的完整遊戲空間	22
圖 3.22 地圖片段採取基因演算法之演化流程	23
圖 3.23 房間容器以遊戲物件的佈局方式作為染色體	24
圖 3.24 動線於房間容器的計算方式，黃色處行經 2 次、紅色處行經 1 次	25

圖 3.25	守衛點指標的遊戲體現情形	25
圖 3.26	阻攔點指標的遊戲體現情形	26
圖 3.27	攔截點指標的遊戲體現情形	27
圖 3.28	巡邏點指標的遊戲體現情形	27
圖 3.29	適應值進行標準化	30
圖 3.30	房間類型 - 戰鬥通道	30
圖 3.31	寶藏房的遊戲物件佈局演化示意	31
圖 3.32	狹路驅逐情境的遊戲物件佈局演化示意	31
圖 3.33	鎮守要道情境的遊戲物件佈局演化示意	32
圖 3.34	將關卡中全部房間容器進行遊戲物件的佈局演化	32



表 目 錄

表 3.1 非法的任務規則定義 13

表 4-1 原始資料之欄位示意 35



演 算 法 目 錄

演算法 1 RewriteSystem1 - 改寫系統（任務語法）	14
演算法 2 RewriteSystem2 - 改寫系統（任務轉換空間）	20



第 1 章 緒論

程序化內容自動生成 (Procedural Content Generation) 在過去就廣泛被應用於遊戲設計領域，其主要目的為增加遊戲內容的隨機性與多樣性。在本文中，我們針對遊戲過程中的遊玩特徵 (gameplay patterns) 進行抽象化，使用程序化生成技術產生帶有意義遊戲關卡內容，藉此消彌或降低因隨機性所產生的不穩定要素，以改善並豐富遊戲體驗。

我們將遊戲關卡的構成劃分為任務 (Missions) 與空間 (Space) 兩種結構後，空間會依照任務結構進行有意義的轉換，接著依照遊玩特徵定義基因演算法 (Genetic Algorithms) 的演化依據。讓玩家在進行遊戲時能夠遵循關卡設計師的劇情脈絡外，亦能夠體驗到有意義且多樣化的遊戲關卡內容。

1.1 研究背景與動機

content here.



1.1.1 迷宮探索遊戲 (dungeon crawl) 類型介紹

content here.

迷宮探索遊戲的歷史

content here.

1.2 研究目的與研究問題

content here.

1.2.1 研究目的

content here.

1.2.2 研究問題

content here.

1.2.3 預計研究貢獻

content here.

1.3 本論文之章節結構

content here.



第 2 章 相關研究

content here.

2.1 含有敘事脈絡的關卡

Jenova Chen 在 [1] 提到，心理學領域的心流理論 (Flow) 應如何被運用到遊戲設計中，且藉由該理論來改善遊戲設計中的..... (編輯中)。

Image is under construction

圖 2.1: 心流體驗應用於遊戲設計



2.2 關卡生成的方法

本節收錄了至今的關卡生成框架與方法。而參考的文獻主要分為兩種類型，分別為 2.2.1 小節的程序化生成任務內容與 2.2.2 小節的程序化遊戲物件擺放。同時收錄數款採取關卡生成技術之遊戲。

2.2.1 程序化生成任務內容

content here.

Mission/Space 框架

Joris Dormans 認為一個完整的關卡需要包含任務與空間二者 [2] [3] [4]；需要有一特定的空間佈局，及一系列需要於此空間中被執行的任務。關卡任務代表玩

家需要按照任務流程，來依序挑戰才能夠完成該關卡；關卡的空間由其地理佈局所組成，或者由與地圖相似的節點網絡所構成。由於任務與空間之間的交錯混雜，導致關卡設計者最終採取簡單卻有效的策略，也就是讓任務與空間同構。雖然同構在設計上不是唯一的選擇，但對於某些遊戲是非常合適的，特別是一具有線性的關卡設計。而 Joris Dormans 亦提出了一種自動關卡設計的方法 [2] [4]，藉由產生一個任務，再利用這個任務去產生適合此任務的空間。舉例來說，關卡設計者透過生成任務的介面來建立任務圖 (mission graph)，玩家必須執行這些任務才能夠完成關卡，接下來將任務轉換為空間，並將任務依序安排至該空間圖 (space graph) 中。設計者接著在地圖添加更細節的內容，直到地圖充滿任務的要素並作為遊戲的關卡。

任務圖注重於任務與玩家的相互關係，表現出玩家距離通關的進度狀況。主要由兩種要件：節點和有向連結線所構成，其中節點再細分為任務、起點與終點；有向連結線再依照兩節點之間的執行先後關係，細分為薄弱條件、強烈條件與抑制。其中，強烈條件或抑制的關係，會導致某些節點無法執行。空間圖直接呈現了關卡的空間結構，且大多數的節點能夠直接表示出玩家目前所在位置。空間圖中的任何節點能透過顏色、字母來表示不同類型。主要亦由兩種要件：節點和連結線所構成。節點細分為場所、鎖和遊戲元素所構成；有向連結線細分為通道、閥、窗、解鎖與上鎖等。

改寫系統 (rewrite system) 由具有左側與右側的規則 (rules) 所組成，能夠將規則中指定的一符號集能夠被另一符號集所取代。改寫規則當中所使用的符號，便是在遊戲中經常會出現一些具有代表性的物件、要素或任務目標等，在字符串表 (alphabet) 中定義以抽象化描述遊戲中的週期性結構 (recurrent construction)。改寫系統能夠套用在構成任務的圖形語法 (graph grammars) 及構成空間的形狀語法 (shape grammars)，二者能夠獨立生成出結果，不過建議能夠將改寫系統套用在任務圖上，使其能夠產生出空間圖。本文提及之任務圖和空間圖是經過改良後的版本，定義其規則時會有些微上的不同，但更能夠體現出遊戲的關卡結構。

Image is under construction

圖 2.2: Mission/Space 框架之主要結構

Dungeon Crawl

content here.

Image is under construction

圖 2.3: Unexplored 的關卡設計

2.2.2 程序化遊戲物件擺放

找 (Ball, 2004, 131-147) 這一篇，介紹複雜系統

原文



In games, as in any complex system, the whole is more than the sum of its parts.

While the active agents or active elements in a complex system can be quite sophisticated in themselves, they usually can be simulated as rather simple models.

Even when the study is about the flow of pedestrians in different environments, great results have been achieved by simulating them with only a few behavioral rules and goals (Ball, 2004, 131-147)

Map Sketches 與 Segments 的演化

Antonios Liapis 開發了策略型遊戲的抽象化地圖生成工具 - Sentient Sketchbook [5]。在 Sentient Sketchbook 中，遊戲關卡設計師能夠以低分辯率、高階抽象的方式來編輯地圖草圖 (map sketches)，構成地圖的瓦磚類型有資源磚、基地磚、不可通行磚與可通行磚等。典型的戰略型遊戲中，每位玩家都必須從隨機選擇的基地開始採集資源以建構戰鬥單位，並利用這些戰鬥單位摧毀敵方基地以完成遊戲。

當設計師編輯地圖時，該工具能夠測試地圖的可玩性（意旨能夠正常進行遊戲）且量化顯示，如果沒有足夠的基地、資源或可連通的路徑，那麼工具提供的遊玩特徵指標將會提示該地圖為不可遊玩的狀態。而這些遊玩特徵指標分別為資源安全性：距離基地僅一格以內的資源磚數量；安全區域：計算基地與敵方基地間的磚總數；探索性：利用洪水填充演算法，計算從基地至敵方基地時，可通行的磚總數。透過用戶當前編輯的地圖草圖，該工具利用基因演算法進行前述等指標，評估適應性函數 (fitness functions) 以解決約束最佳化 (constrained optimization) 等問題，來產生出更多意想不到的地圖輸出結果。

後續的研究中，Antonios Liapis 將基因演算法調整為兩階段演化，第一階段演化為地圖草圖演化，第二階段為地圖片段 (map segments) 演化 [6]。地圖片段的結構類似於地圖草稿，由 $N \times N$ 的瓦磚所構成，瓦磚的種類能夠像是空磚、牆、連接處、出口、怪物或寶箱等，其中連接處是為了讓地圖片段彼此能夠接合以填滿地圖。利用地圖草稿所轉化成的初始地圖片段可作為演化用的胚胎 (embryogeny)，於此階段定義的牆、連接處會呈顯穩定狀態，不隨著演化過程而改變，其餘瓦磚有機會由空磚突變為怪物、寶箱或牆，反之亦然。並探討不同的目標函數與胚胎，如何影響的圖片段的最佳解與外觀。

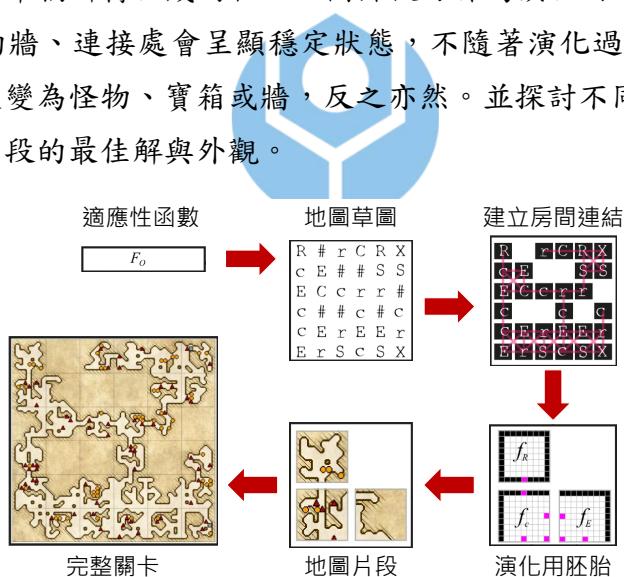


圖 2.4: Antonios Liaps 提出的兩階段式關卡演化

2.3 使用關卡生成技術的遊戲

content here.

遊戲 - Unexplored

Unexplored 是由 2.2.1 小節提及之框架作者 Joris Dormans 所製作，Joris Dormans 基於 Mission/Space 框架並提出改良的即時型戰鬥 roguelike 遊戲（若再細分可歸類於 roguelite 遊戲類型）。在市面上普遍的 roguelike 遊戲中，最常見的關卡生成策略即是視地下城玩家起點位置為樹之根，再以此根持續添加生成或預先設計的

Joris Dormans 稱改良後的方法為環狀地下城生成 (cyclic dungeon generation) ，在每一層的地牢中...

2.3.1 遊戲 - Spelunky

[7] [8]



2.3.2 遊戲 - The Binding of Isaac

content here.

2.4 基因演算法

content here.

第3章 研究方法

本論文中，我們仍保持 Joris Dormans [2] [4] 與 Antonios Liapis [5] 為求遊戲設計過程抽象化與高階化的訴求。將「Mission/Space 框架」與「Multi-segment 演化」兩種關卡生成方法結合並予以改良，保留了前者追求的遊戲進程之順序性，後者帶來穩定且多樣化的遊戲內容，希冀藉此提升整體遊戲體驗、相輔相成。

圖 3.1 為系統的整體流程圖。遊戲設計師能夠透過宏觀的觀點來構築遊戲體驗流，將遊玩特徵拆分成多項規則，利用生成語法及改寫系統生成出任務圖，圖 3.1 a。依照任務圖中對應的終端節點 (terminal nodes) 轉換為事先建構完成的房間容器，並得到尚未包含遊戲物件的遊戲地圖，圖 3.1 b, c。接下來，針對動作冒險遊戲我們提出了數項評估遊戲性的適應性函數，藉由基因演算法的演化流程，令各房間遵守適應性函數的限制，以得到符合訴求的最適解，圖 3.1 d, e, f。

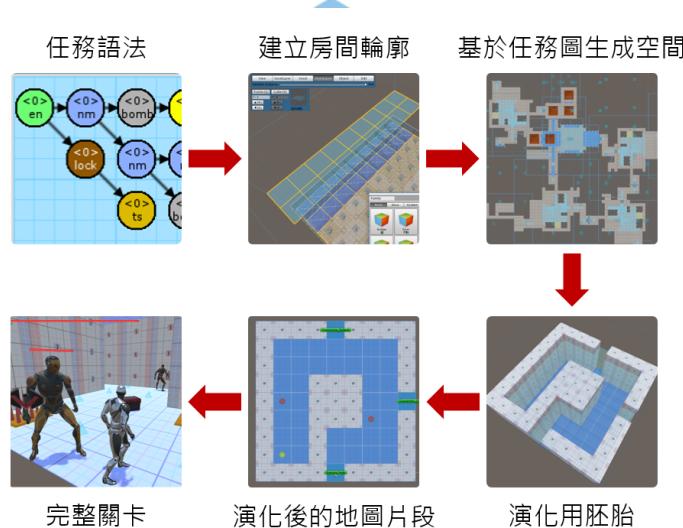


圖 3.1: 本論文提出系統之流程圖

3.1 任務語法

我們基於 Joris Dormans 提出之 Mission Grammars 概念，進行實作與改良出 *Dungeon Generator* 工具，這項工具佈署在 Unity Engine 上。遊戲設計師能夠藉由 *Dungeon Generator* 工具進行任務語法的建置，並執行改寫系統以輸出任務圖，進一步利用任務圖產出遊戲關卡空間。

在任務語法的設計階段，我們參考 2.3.2 小節所提及之遊戲 *The binding of Isaac* 的關卡地圖，分析其遊戲進程結構，構想期望的任務圖並將觀察其遊玩特徵，接著拆分成任務語法規則使用改寫規則產生出近似結構的任務圖。

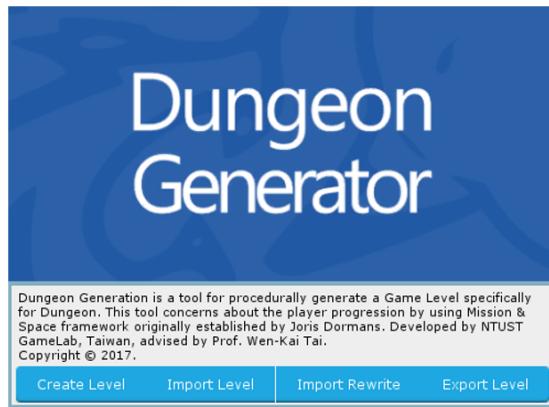


圖 3.2: Dungeon Generator 工具



3.1.1 建立任務符號表

在本小節，會以 *The binding of Isaac* 遊戲關卡作為範例，解析其遊玩特徵並作為任務語法的節點。

解析遊玩特徵

在任務語法中，任務節點可表示為一項事件、挑戰、動作或遊戲物件等。藉由觀察 *The binding of Isaac* 的遊戲關卡後，能夠發現房間擁有固定類型，並且對於房間之間進行程序化的排列組合。然而固定的房間類型便是我們所尋找的節點種類，藉此歸納出「入口 (entrance)、出口 (goal)、魔王房 (boss)、一般房 (normal)」外，以及一定機率會出現的稀有特殊房間「秘密房 (secret)、商店 (shop)、寶藏房 (treasure)」。這些稀有房間帶來了一些制式的事件，在進入秘密房前必須使用炸彈將牆壁給破壞的事件 (bomb)，以及從秘密房取得的鑰匙能夠進行解鎖 (lock)，開通進入商店或寶藏房的權限。而前述的遊玩特徵將轉化為終端節點 (terminal nodes) 的圓形符號形式，表示較為詳盡的任務內容；另外一種為方形符號的非終端節點 (non-terminal nodes)，相較於終端節點更為高階、抽象化，在此我們定義出起始節點 (Start)、一般房 (Normal) 與特殊房 (Special) 等，其應用方式將在下一小節 3.1.2 中介紹。

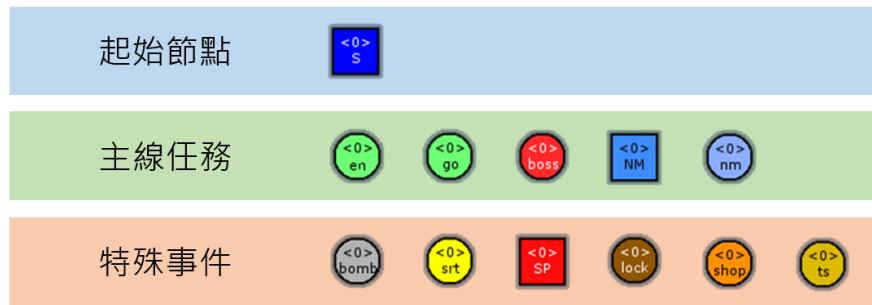


圖 3.3: 利用遊玩特徵建立任務符號表之範例

3.1.2 建立任務規則

任務規則分為左側規則及右側規則，二者皆為圖形語法 (Graph Grammars) 所構成，左側規則為被取代方、右側規則為取代方，在任務圖中若有子圖符合任務規則的左側，將會依照流程進行替換改寫動作至規則右側。為了方便管理與分類不同性質的規則，在 *Dungeon Generator* 中定義一任務語法包含了多個任務規則群組，各自群組底下有複數個任務改寫規則。規則被賦予使用數量上的條件限制，分為使用上限次數與下限次數，上限次數表示完整的改寫系統運行過程，最多能夠套用該規則的次數上限，多為了因應部分遊玩特徵的獨特性或稀有性；反之，下限次數表示套用該規則的次數下限，能用於改寫系統的疊代結果已趨於穩定、無剩餘任何非終端節點的情形下，作為持續進行改寫系統的強制約束條件。

線性任務規則

玩家在進行遊戲時，會有一條的主要任務流程、劇情安排，圖 3.4 設計了三道任務規則，分別為主線任務、魔王房與一般房。主線任務規則中，系統會從預設起始節點 (starting node) 開始進行改寫，並轉換為由終端節點起點 (entrance) 與終點 (goal) 所構成的線性任務，其中兩節點之間由多個非終端節點的一般房 (Normal)，而非終端節點的一般房會倚賴其它規則進行後續替換。魔王房規則中，規則左側中定義了必須為非終端節點的一般房 (Normal) 與終點 (goal) 相連，並改寫該一般房為魔王房 (boss)。一般房規則中，會將任務圖中的剩餘非終端節點的一般房 (Normal) 替換為終端節點的一般房 (normal)。

結合後續第 3.2 章節 - 空間建構，並以主線任務相關規則生成任務圖後，便可將關卡輸出近似於圖 3.5 的空間。

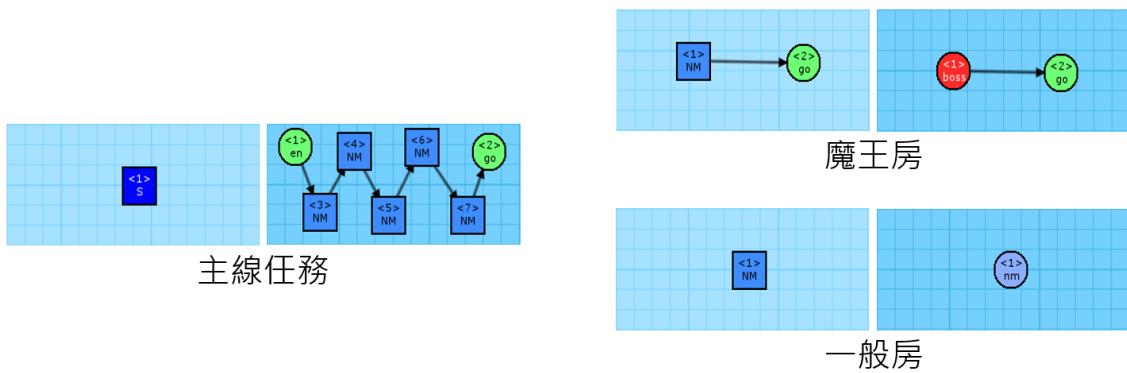


圖 3.4: 線性任務規則範例，建立主線任務

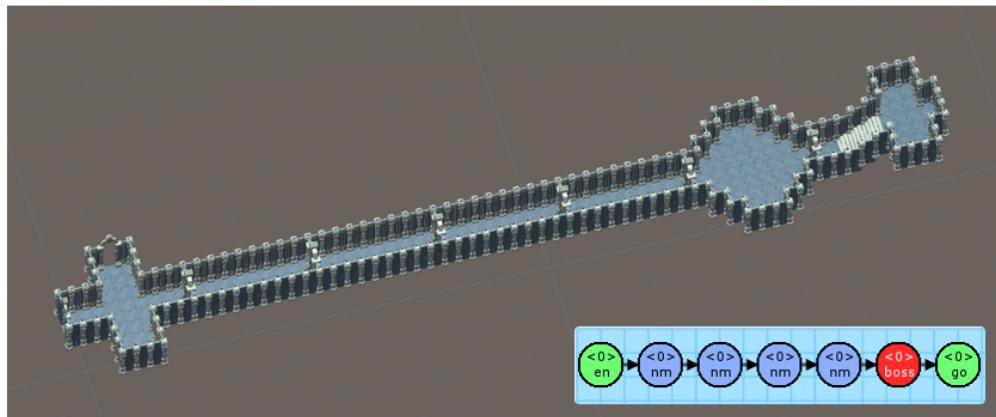


圖 3.5: 建立主線任務的預期任務圖與空間預覽

非線性任務規則

圖 3.6，為求遊戲內容的多樣性，非終端節點的一般房 (Normal) 將有機會轉換為其它種類的遊玩特徵，圖 3.6 定義了特殊房間的配置工作。配置特殊房規則中，非終端節點的一般房 (Normal) 與任一節點 (? - Any) 相連的圖形語法，會被轉換為規則右側的非線性任務圖形語法，當中的秘密區域 (secret) 能夠獲取鑰匙，並有兩道需要由炸彈才能夠突破的牆壁 (bomb) 與其它區域阻隔，此外延伸出特殊房 (Special) 需另外定義規則使之轉換為終端節點。由於這項規則屬於較為稀少的任務類型，在套用數量上將有所限制。在商店規則與寶藏房規則中，規則左側中定義了非終端節點的特殊房 (Special) 能夠被改寫為鎖 (lock) 與商店 (shop) 或寶藏房 (treasure) 相連的遊玩特徵。

圖 3.7，建立支線任務規則能讓指定的節點額外分支出支線任務，替換後的非終端節點一般房 (Normal) 便能再經過其它的規則，取代成不同的遊玩特徵。

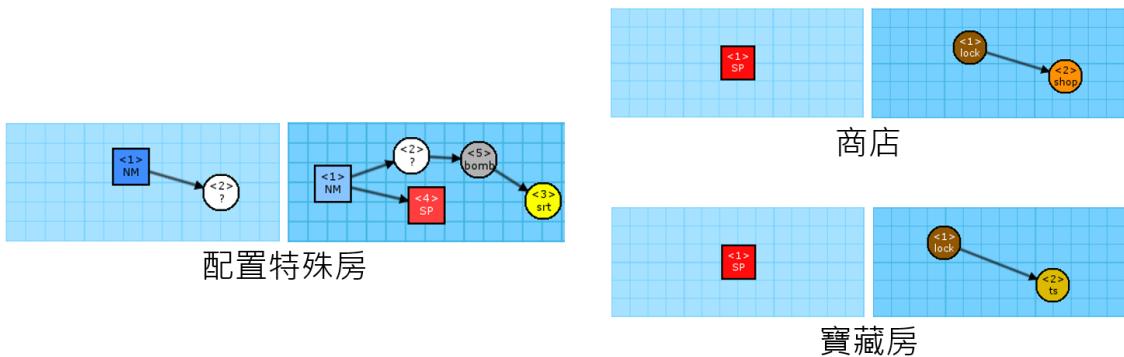


圖 3.6: 非線性任務規則範例，設置特殊房

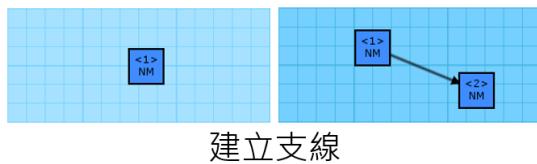


圖 3.7: 非線性任務規則範例，建立支線任務

圖 3.8，為使玩家進行遊戲能夠提前遇見鎖 (lock)，當下卻找不到鑰匙的遊玩特徵，進而思考、尋找開鎖的方式，便利用鎖向前規則將鎖的分支一同往前挪動。房間攤平規則，為避免單一房間與過多的房間連通，將部分房間進行移前推挪。

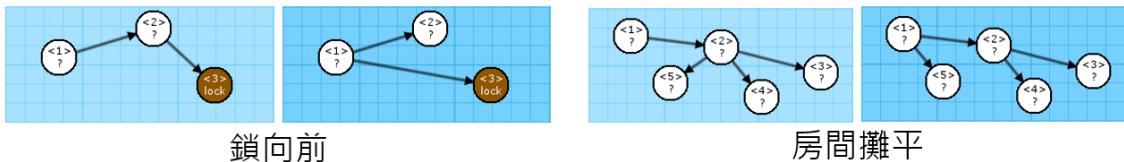


圖 3.8: 非線性任務規則範例，鎖的提前出現與房間攤平

排除非法規則

設計規則時，*Dungeon Generator* 將排除不合法的九項設計原則，非法的設計特徵會導致改寫系統出現錯誤，例如：改寫系統陷入無限循環或任務圖破碎化。在表 3.1 所列舉之條件於某些情況中，可能彼此會同時符合多項非法條件，系統將依照判斷順序擇一輸出。

Image is under construction

圖 3.9: 於 Dungeon Generator 工具中，若規則為非法狀態將不予以生效

表 3.1: 非法的任務規則定義

非法規則之標籤	標籤的狀況描述
LeftMoreThanRight	當左側的節點超過右側的節點數量時，進行改寫系統會使左側無法對應到右側的節點產生遺失的情形。若這些節點原先已有與其它非規則內節點連接，將會導致該連接資訊遺失，有機會造成任務圖破碎。
EmptyLeft	左側為空將無法進行子圖搜索，因此左側節點必須至少一個節點。
IsolatedNode	孤立的節點將會導致任務圖破碎。
IsolatedConnection	孤立的連接線無法正確表示其連接資訊，將無法正常進行改寫系統。
ExactlyDuplicated	若左右規則同構將導致改寫系統陷入無限循環。
MultipleRelations	兩兩節點間不可有超過一個的連接關係，不論是同向連接線或反向連接線皆會導致改寫系統無法正常運作。
CyclicLink	任務圖的定義中，任務應嚴格遵守任務間之順序性，若有循環結構將會使玩家迂迴停滯。
OrphanNode	若有圖形語法含有兩個以上的根結點，便無法正確定位出任務起點。
OverflowedAnyNode	當右側規則使用 Any 節點時，其對應到左側索引值的節點亦必須為 Any 節點。反之，左側規則使用 Any 節點將不在此限。

3.1.3 產生任務圖

根據 Joris Dormans [2] 提出的方法並修改至符合我方實驗需求，歸納出演算法 1。任務語法的改寫系統運行時以深度優先搜尋法 (Depth-first search) 遍歷整個任務圖，過程中遵循以下步驟順序。第一步驟，源節點會從任務規則集合中過濾

出相符的規則，若有多項規則同時符合替換的條件時，系統會基於它們的關聯權重 (relative weight) 從複數個規則中依照採輪盤法 (roulette wheel selection) 選擇出一項任務規則稱之為匹配規則 (matched rule)，並從源節點進行與匹配規則的改寫替換，符合的條件為定義遍歷中的源節點作為新圖，而新圖存在一個子圖與匹配規則的左側之圖形語法同構，同構的參考基準為節點的符號種類，在確認子圖同構的同时，有關連的節點會標記與匹配規則一致的索引值。第二步驟，將任務圖含有索引值的節點，其相連的連接線移除。第三步驟，任務圖含有有索引值的節點取代成匹配規則右側的等價節點。第四步驟，將匹配規則右側中沒有與左側等價的節點添加至任務圖中。第五步驟，依照匹配規則右側的連接線，以相同方式放入任務圖中。最後一步驟，將任務圖中的索引值移除。

在 3.1.2 小節提及，完整的任務圖必不包含任何的非終端節點，倘若經過多次疊代改寫仍有非終端節點存在，關卡設計師必須修改任務語法其規則，使之能夠完整轉換成全終端節點的任務圖，方可進入下一階段的空間轉換。

Algorithm 1 RewriteSystem1 - 改寫系統（任務語法）

Input:

root is current pointer in the mission graph.

rules is a rule set of mission grammars.

seed is the random seed.

Output:

```
1: if matchedRule is found from root then
2:   matchedRule = FindMatchs(root)
3:   移除與 matchedRule 相符子圖的連接線
4:   將相符子圖的節點，依照對應索引值替換成 matchedRule 的規則右側
5:   將 matchedRule 剩餘的節點填充至任務圖中
6:   依照 matchedRule 的連接情況，移轉到任務圖中
7:   移除節點的索引值相關資訊
8: end if
9: for all child such that child ∈ children of root do:
10:   RewriteSystem1(child, rules, seed)
11: end for
12: return root
```

(備註，此處會補上 FindMatchs 在我方實驗環境中如何實現，目前還在構思要

如何撰寫。)

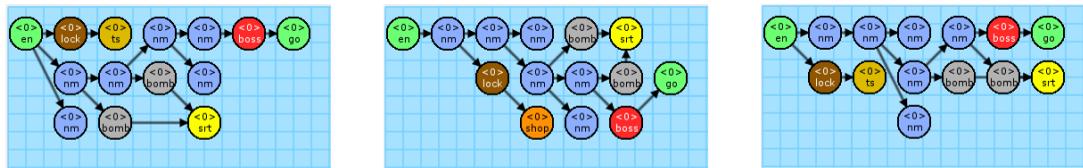


圖 3.10: 最終輸出的任務圖



3.2 空間建構

Joris Dormans 於文獻中提到為二維空間的範例 [2] [4]，我們的實驗環境以三維空間為主。在空間語法中將直接構築遊戲的基礎房型，但不設置怪物、寶箱或陷阱足以直接影響遊戲性的遊戲物件，如圖 3.11。此外，我們希望空間中的遊戲物件能夠有意義的自動化配置，即在設計空間語法的流程中，忽略絕大部分的遊戲物件配置，直到 3.3 節提出之方法達成。



圖 3.11: 影響遊戲性的遊戲物件，由左至右分別為敵方、寶箱與陷阱

3.2.1 基礎結構

我們對於空間語法做了修改以利實驗環境建置。圖 3.12 所示，在一個關卡 (level) 中包含數個房間容器 (volumes)，每一房間由不定數量的房間塊 (chunks) 組成，且房間塊固定以 $9 \times 9 \times 9$ 個長方體體素 (voxels) 所構成，每一體素的大小為 $3 \times 2 \times 3$ 。

此外，每一個體素被分為九種方向，分別為正四方、斜四方與中心點。不同的裝飾物會依照其特性決定能夠放置的方位，如牆壁放置於正四方；牆壁柱放置於斜四方；地面或階梯則放置於中心點。在一個體素中，多個裝飾物是能夠同時並存的，例如在其放置地面、一道牆壁、兩道牆壁柱。在本次實驗環境中，我們將會使用圖 3.13 中，「地面、階梯、牆壁、牆壁柱、門」共五種裝飾物進行房間容器的建置工作。



圖 3.12: 房間容器的建構情形，可觀察其由體素單位所構築而成

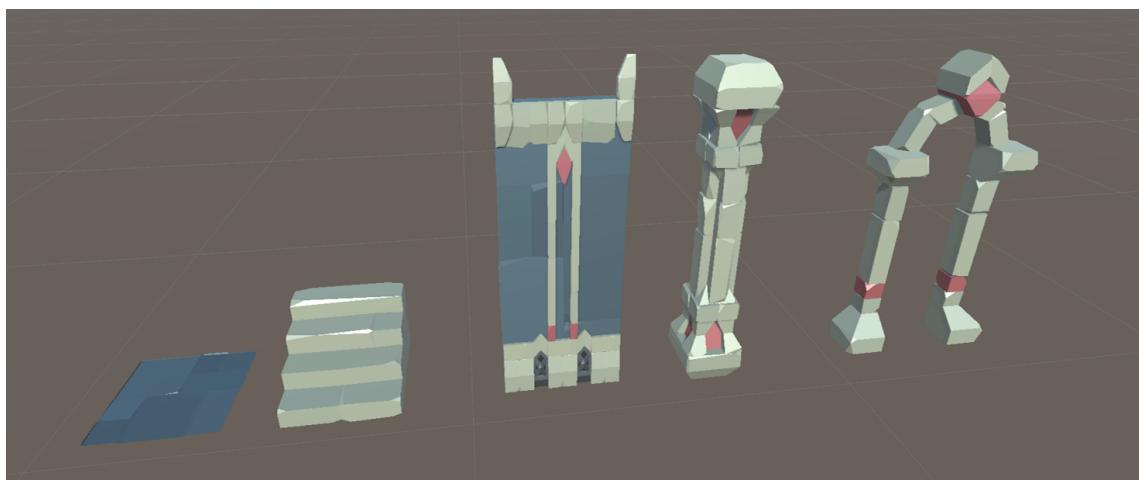


圖 3.13: 房間容器時所使用的五種裝飾物

3.2.2 端點識別物

逐一設計出各式各樣的房間容器後，必須正確地將個空間相互連接。參考 Joris Dormans 的空間語法 [4]，我們為體素型的房間容器添加端點識別物 (connections)，而端點可再細分為入口 (entrance) 與出口 (exit)，依照遊戲設計師需求，能夠自行擴充出口的類型，可為複數個出口識別物。在一房間容器中，最多僅能擁有一個入口識別物，而出口識別物的數量不在此限，且二者之數量總和必大於零。倘若空間當中沒有一個以上入口識別物，便會從現有的出口識別物中隨機挑選一與入口識別物相同功能執行之。圖 3.14 中描述了端點識別物的使用情境，入口的箭頭朝向空間內部，而出口的箭頭會朝向空間外部。

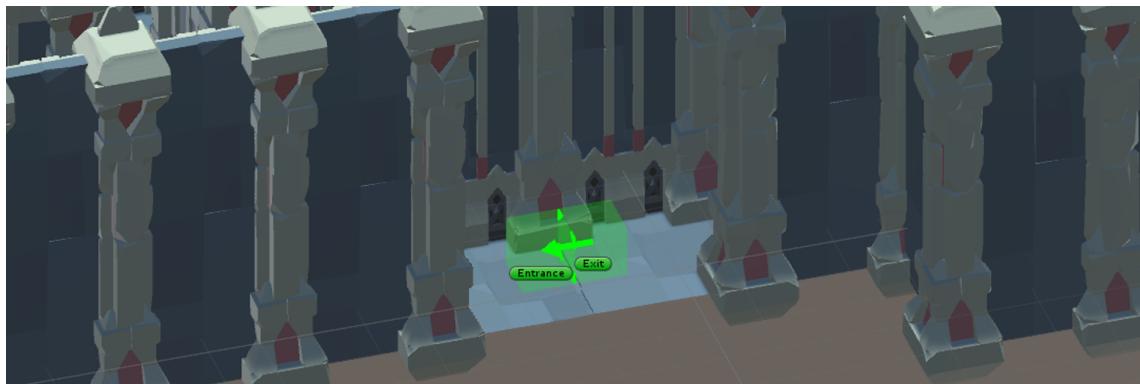


圖 3.14: 端點識別物的實際使用與串接情形

3.2.3 房間容器類型

延續第 3.1 節任務語法所定義的任務終端節點，根據依照節點性質設計房型，區分出數項不同房型的空間規劃。

入口、出口、魔王房



入口擁有一個入口與三個出口，為玩家重生的起始房間。出口擁有一個入口，為遊戲關卡結束的房間。魔王房擁有一個入口與一個出口，玩家會在此房間遭遇強大敵人。

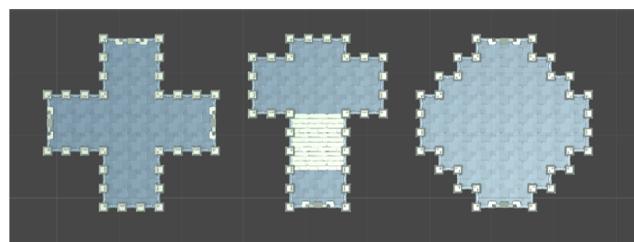


圖 3.15: 房間類型 - 由左至右為入口、出口與魔王房

通道

通道共有四種，隨著端點數量有等同的出口數量，因沒有入口識別物，便能夠以任一出口作為入口，搭配房間容器旋轉能達到共計 11 種房型同構。

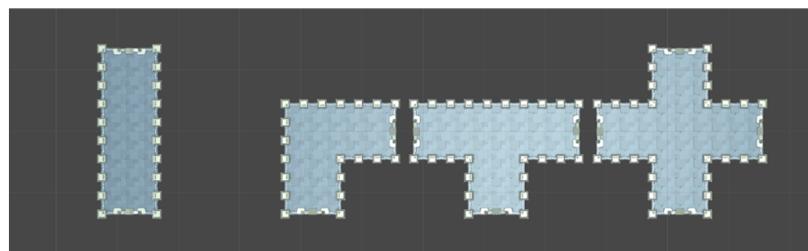


圖 3.16: 房間類型 - 由左至右為 I 型、L 型、T 型與 + 型通道

石牆、秘密房、鎖、商店、寶藏房

石牆擁有一個入口及一個出口，而出口的相同位置有石牆識別物。秘密房擁有一個入口，玩家能在該空間取得鑰匙識別物。鎖空間擁有一個入口及一個出口，在出口的相同位置有鎖識別物，玩家必須取得鑰匙方可通過。商店與寶藏房各擁有一個入口，玩家能夠在該房間觸發特殊事件。

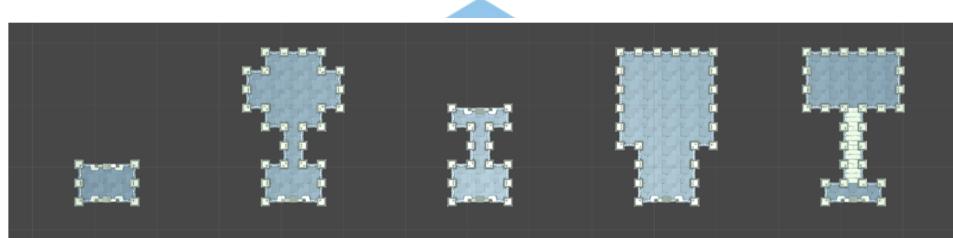


圖 3.17: 房間類型 - 由左至右為石牆、秘密房、鎖、商店與寶藏房

死路

在第 3.2.4 小節中，為了要封閉剩餘的端點識別物所造成的空間開口，便需要無特殊功能的單一入口空間容器。

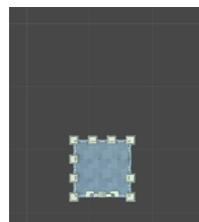


圖 3.18: 房間類型 - 死路

3.2.4 從任務圖轉為遊戲空間

Joris Dormans 提到任務圖與空間圖應當是非常近似，甚至是二者同構 [2]。本小節將解釋 *Dungeon Generator* 工具運行「將任務轉為空間的改寫系統」的流程，以及如何搭配任務圖的語法分析器。其一，藉由「建造方式表」能將任務圖轉換成有意義拓撲關係的遊戲空間；其二，利用「空間替換表」把前述遊戲空間不足之處補足。完成上述改寫系統進程後，便能夠輸出帶有任務邏輯的遊戲空間。演算法 2 中，系統會遵循第 3.1.3 小節產生的任務圖，以深度優先搜尋法遍歷任務圖，經過的源節點遍從建造方式表中挑選出一個房間容器，逐一將各個終端節點轉換為實際空間，接著透過建造方式表，將場面上剩餘的端點識別物依序替換成對應房間容器，以完成遊戲空間之建置。

Algorithm 2 RewriteSystem2 - 改寫系統（任務轉換空間）

Input:

graph is current *MissionGraph*.

seed is the random seed.



Output:

```
1: for all node such that node  $\in$  graph do:                                ▷ Instrcution
2:   volume = GetVolumeFromInstructionTable(node, seed)
3:   isSucceed = SetVolumeToSpace(volume, seed)           ▷ return true or false
4:   if isSucceed is false then return false
5:   end if
6: end for
7: for all marker such that marker  $\in$  ConnectionsInGraph do:      ▷ Replacement
8:   volume = GetVolumeFromReplacementTable(marker, seed)
9:   isSucceed = SetVolumeToSpace(volume, seed)           ▷ return true or false
10:  if isSucceed is false then return false
11:  end if
12: end for
13: return true                                         ▷ Successfully generated a game space
```

建造方式

為從已生成完畢的任務圖進而衍生出遊戲空間，在創建的房間容器會對應一任務符號表當中的終端符 (terminal symbols)，稱作為建造方式 (building instruction)。本小節會利用第 3.1.1 小節的任務符號表，過濾出終端節點並逐一對應至第 3.2.3 小節的房間容器，見圖 3.19。每一個終端節點都必須對應一個以上的房間容器，房間容器不允許被重複使用在不同的終端節點上。

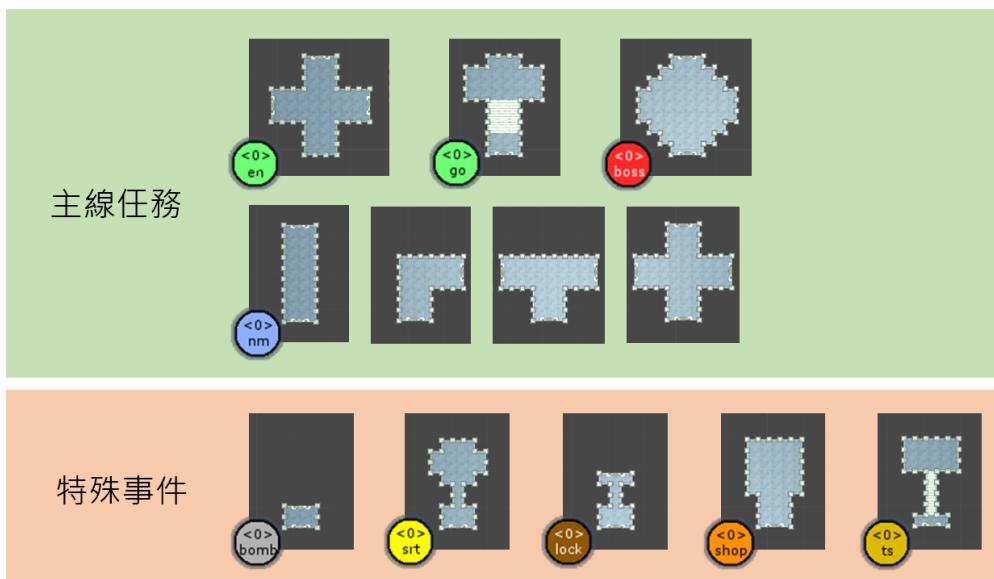


圖 3.19: 建造方式，任務終端節點與房間容器對應之範例

基於建造方法的對應紀錄，給予一亂數種子能夠確保能夠對應到一固定空間。當終端節點對應到複數房間容器時，亂數種子將影響從多個房間容器之間的順序。若生成過程中，該房間容器擁有多個出口，亦是參照亂數種子決定調用出口生成之順序，見圖 3.20。

剩餘空間替換

在圖 3.20 可觀察到，經過建造方式的遊戲空間仍會遺留部分的端點識別物，為了將其延伸或予以封閉，本小節將進行替換 (replacement) 以達目的。第 3.2.2 小節，介紹端點識別物時，曾提及到遊戲設計師能夠自由擴充出口識別物的種類。在此為簡化範例，我們採用預設之單一出口識別物，而這項出口識別物將對應圖 3.18 的死路，若增加對應的選項進而創造出主線外的分支結構，見圖 3.21。

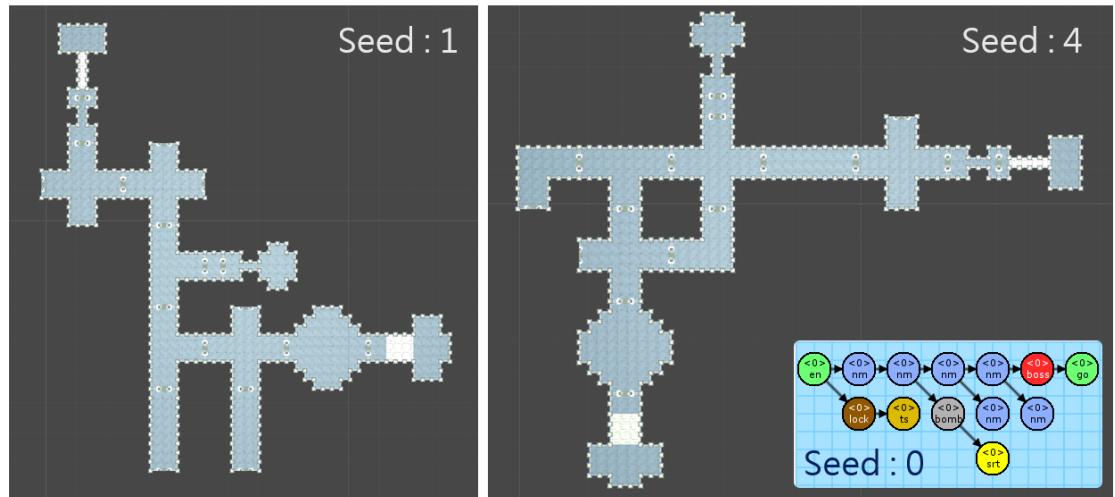


圖 3.20: 已疊代生成的任務圖，不同亂數種子轉換為對應結果的遊戲空間

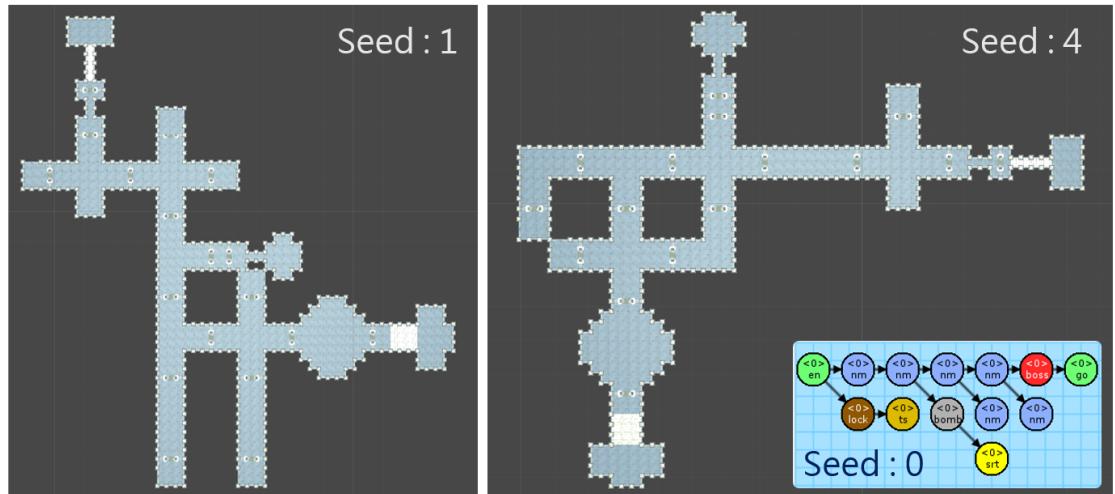


圖 3.21: 經過剩餘空間替換後的完整遊戲空間

3.3 地圖片段

為了將第 2.2.2 小節提及的程序化遊戲物件擺放技術付諸實現，構築出遊戲內部的複雜系統，讓玩家體驗到突現型 (emergence) 的遊玩機制，我們將延續上一節的遊戲關卡結構，參考 Antonios Liapis 提出的地圖片段演化方法 [6]，進行改良以合適我方實驗環境。

圖 3.22 介紹地圖片段的演化流程。第一步驟，基於 3.3.1 小節中定義的基因結構，產生初始父母代族群時，讓全部的基因先預設為空磚；第二步驟，透過適應性函數計算各個體的適應值 (fitnesses)，完整的適應性函數在 3.3.2 小節中說明；第三步驟將會從族群中挑選最優異的兩個父母染色體，高機率進行交配，若無進行交配將會將子代沿用父母代的基因；第四步驟有低機率讓衍生的子代進行突變；第五步驟以新的衍生子代取代舊有的父母代族群；第六步驟會檢查是否達到終止條件，若尚未滿足終止條件，便會回到第二步驟，直到輸出最適解。前述之交配、突變事件的機率與其方法屬於實驗變因，將在第 4 章進行定義與相關釋義。

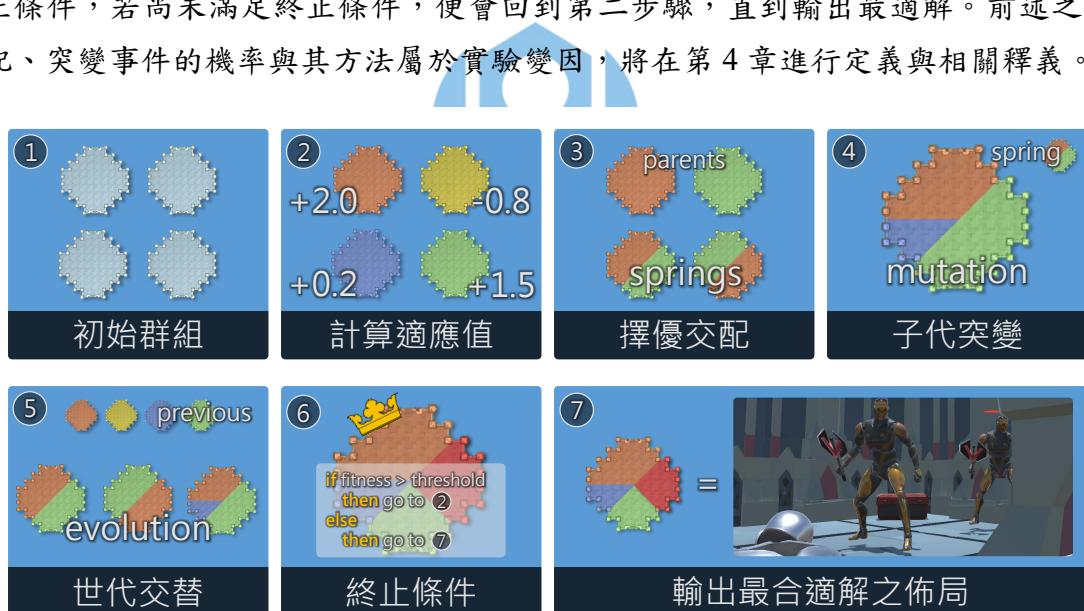


圖 3.22: 地圖片段採取基因演算法之演化流程

3.3.1 基因的儲存結構

圖 3.23 a 是遊戲物件的表型 (phenotype)，為基因演算法中的個體單位，房間容器能夠存放數種遊戲物件，分別為空磚 (Empty)、敵人磚 (Enemy)、寶箱磚 (Treasure) 與陷阱磚 (Trap)，圖中帶有編號的位置意指著該座標能夠擺放遊戲

戲物件，且玩家角色能夠在該座標通過；遊戲物件的擺放位置為個體的染色體 (chromosome)，以一維陣列之基因序列表示，見圖 3.23 b；基因序列中的基因，會存放該座標的相對位置與遊戲物件類型，見圖 3.23 c。同一個房間容器會擁有許多不同的個體，這些個體的集合便是族群 (population)。

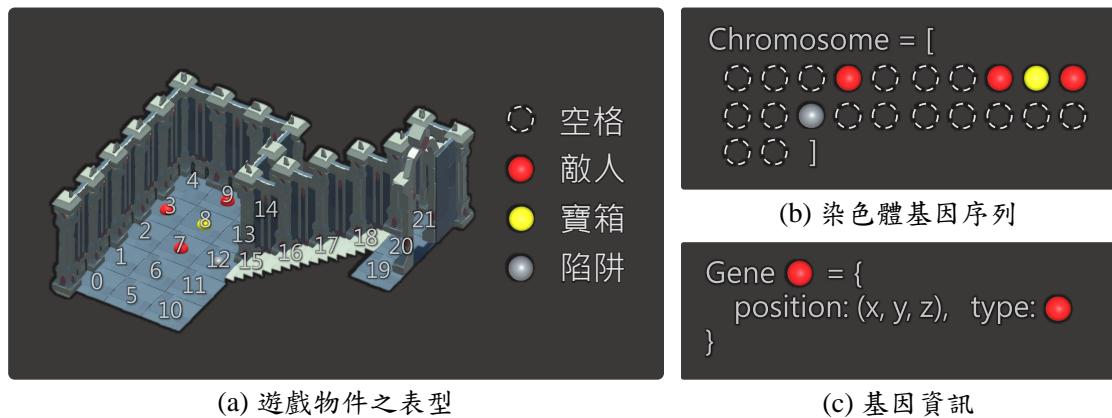


圖 3.23: 房間容器以遊戲物件的佈局方式作為染色體



3.3.2 演化之適應性函數

動作冒險遊戲 (A-AVG)、動作角色扮演遊戲 (A-RPG) 等類型遊戲，多可見一些制式化遊戲物件的搭配組合，Alexander Baldwin 認為遊戲物件組合的細觀特徵 (Meso-Patterns) 需要有檢測其品質的方法 [9]。我們嘗試汲取多項遊玩特徵並參數化公式，作為評估關卡品質的指標之一。

於前處理階段時，圖 3.13 中顯示的地面、階梯裝飾物屬於可以通行的瓦磚單位（請參考圖 3.13 a, b），使用 A-Star 搜尋演算法建立行走空間，並搜索入口至多個出口的最短路徑，凡經過的座標稱作為空間動線 (MP) 之一，並將其權重值 (mp) 增加 1，空間動線為多項指標關鍵性的參考依據。倘若遇到無法計算最短路徑的情形，便不存在空間動線，屆時該房間容器將不能夠套用相關適應性函數。

守衛點 (Guard)

方程式 3.1 為守衛點的適應性函數。為體現出敵人會保衛寶箱 (Treasure) 與出口 (Exit) 的現象，計算敵人 (E_i) 與為守護對象的遊戲物件 (O_j) 之間的距離，倘

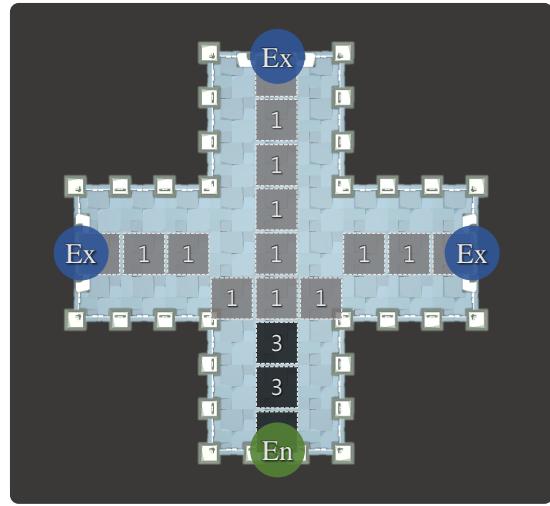


圖 3.24: 動線於房間容器的計算方式，黃色處行經 2 次、紅色處行經 1 次

若距離愈近則帶來的影響力愈大。

$$f_{grd} = \sum_{i=1}^M \sum_{j=1}^N \frac{1}{dist(E_i, O_j)}, O_j \in \{\text{Treasure}, \text{Exit}\} \quad (3.1)$$



圖 3.25: 守衛點指標的遊戲體現情形

阻攔點 (Block)

方程式 3.2 為阻攔點的適應性函數。敵人會專注於阻攔玩家繼續前進，迫使玩家與其發生衝突。 f_{blk} 為加總 M 個敵人於空間之動線權重 (e_i)，倘若敵人並未

落在動線上，權重則為 0。圖 3.26 為阻攔點指標的實際遊玩情形，灰色區塊為系統計算出的空間動線的瓦磚，敵人將會被設置於動線之上阻攔玩家前行。

$$f_{blk} = \sum_{i=1}^M e_i \quad (3.2)$$



圖 3.26: 阻攔點指標的遊戲體現情形



攔截點 (Intercept)

(編修中)

方程式 3.3 為攔截點的適應性函數。與阻攔點近似，但敵人會避免被配置在空間動線上，為求快速追擊玩家為目的，將圍繞在空間動線附近。各敵人 (E_i) 越接近空間動線各點 (MP_j) 時影響愈大，且動線權重 (mp_j) 亦會影響加權程度。

$$f_{itc} = \sum_{i=1}^M \sum_{j=1}^N \left(\frac{1}{dist(E_i, MP_j)} \times mp_j \right), E_i \neq MP_j \quad (3.3)$$

巡邏點 (Patrol)

(編修中)

確保各敵人擁有足夠的空間能夠進行移動。將計算敵人 (E_i) 與指定半徑 (r) 內的座標數量 (P_j) 總值，當中並不包含不可通行的牆壁等類型。於本次實驗中，

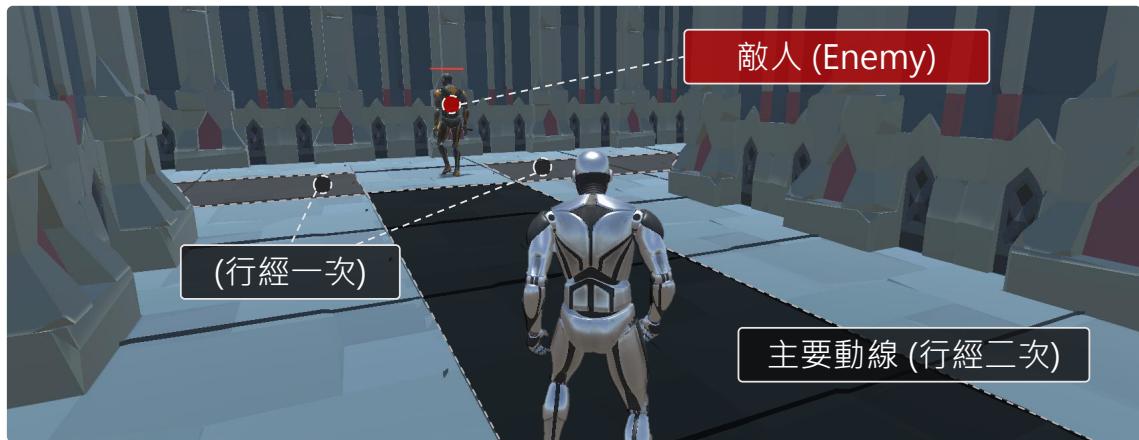


圖 3.27: 擋截點指標的遊戲體現情形

我們將採用 $R = 3$ 作為實驗範例，該數值可由遊戲設計師決定。

$$f_{ptl} = \sum_{i=1}^N \left(d_i \times \sum_{j=1}^M \text{count}(E_i, P_j) \right), d_i = \begin{cases} \frac{1}{2^i}, & \text{if } i \neq N \\ \frac{1}{2^{i-1}}, & \text{if } i = N \end{cases} \quad (3.4)$$
$$\text{dist}(E_i, P_j) \leq r, P_j \notin \text{wall}$$

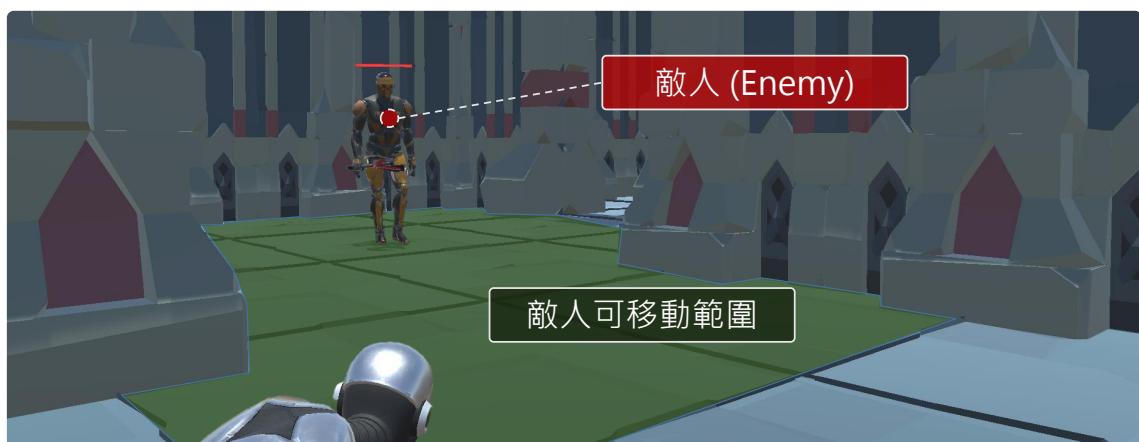


圖 3.28: 巡邏點指標的遊戲體現情形

支援點 (Support)

(編修中)

敵人(E_i, E_j)之間擁有一定度的護援關係，當敵人彼此的距離愈低其影響程度越大，同時該敵人(E_i)必須遠離動線(MP_k)。

$$f_{sup} = \frac{1}{N} \sum_{i=1}^N \left(\frac{1}{N} \sum_{\substack{j=1 \\ j \neq i}}^N \frac{1}{dist(E_i, E_j)} + \frac{1}{M} \sum_{k=1}^M \frac{1}{dist(E_i, MP_k)} \right) \quad (3.5)$$

陷阱點 (Trapped)

$$f_{trp} = \quad (3.6)$$

3.3.3 制衡取向的適應性函數

於3.3.2小節所提及的適應性函數中，多數的函數有著一項共通點，便是隨著符合遊玩特徵的物件數量與適應值呈現顯著正相關，隨著演化代數的提升至終導致房間容器充斥著大量的遊戲物件。然而這樣的遊玩體驗在絕大多數都是不可行的，因此需要針對現有適應性函數所衍生出的特性，設計抑制其得分隨著數量級的成長。

方程式3.7控制該房間容器出現的遊戲物件總數。房間容器的物件總數上限為 M 、下限為 N ，若遊戲物件總數 O 於數量範圍內可得1分，反之得0分，本適應性函數的權重固定為1。

$$f_{dst} = \begin{cases} 1, & \text{if } N \leq O \leq M \\ 0, & \text{if } O < N \text{ or } O > M \end{cases} \quad (3.7)$$

3.3.4 負數權重的適應性函數

在3.3.2守衛點指標的設計中，當權重為正數時能夠體現出「需要被保護的物件，其周圍分配敵方單位進行守衛」的遊玩特徵，且在多個物件的情形下儘可能平均分配敵方單位；反之，負數權重便會體現出「數量愈不平均的守衛情形」。緣故為設計守衛點指標時，並無針對寶箱的數量進行控管而導致。

3.3.5 多項適應性函數合併

在設計適應性函數初期，為求多個適應性函數相互牽制，進而求得限制條件的最佳解。而初期函數會力求場上所有的敵人必須盡可能符合各項指標，舉例來說，在某一房間容器的設定中，我們將守衛點、阻攔點的權重調整至 1，其餘指標將不採計得分即不調整權重，倘若場面上存在著敵人 A 與敵人 B，他們會同時被設置在動線上且一定距離內會有作為守護對象的寶箱物件。但對於遊戲設計師在某些情形下，敵人 A 與敵人 B 僅需要分別符合守衛點、阻攔點指標，才是理想中的游玩特徵。因此，我們定義「首次出現相關游玩特徵的得分幅度最顯著」，並依照出現數量逐漸下降成長幅度且逼近於 1，呈現近似於對數函數圖形的曲線，因此上述兩種情形都有機會成為演化過程中的最適解。取代原先的線性成長關係，進行數值的標準化，強制將值域規範至 $[0, 1]$ 以求最後適應性函數加權總分時，不偏袒任何適應性函數以達到公平基準，並於第 4.4 節進行相關驗證。

方程式 3.8 中定義房間容器（個體）的品質，採計共 M 種適應性函數，第 i 項適應值為 f_i 並經過標準化 $Normalized(f_i) = [0, 1]$ ，依照各函數得分與其權重值 $w_i = [-1, 1]$ 加權後加總得到 f_{all} 。而標準化由個體分數 f_i 除以該世代最高得分 $f_i \cdot max$ 開 c 次方根。

$$f_{all} = \sum_{i=1}^M (Normalized(f_i) \times w_i) \quad (3.8)$$

$$Normalized(f_i) = \left(\frac{f_i}{f_i \cdot max} \right)^{\frac{1}{c}}$$

圖 3.29 中顯示了方程式 3.8 依照不同的非零自然數常數 c 進行適應值的標準化，在此圖中定義 $x \cdot max = 10$ 方便後續的解釋工作。 y_1 為原始分數除以原始分數最大值之標準化，初期的成長幅度為 $x_{12} - x_{11} = 0.1000$ ，隨著原始分數的增加並不影響後續的成长幅度 $x_{14} - x_{13} = 0.1000$ 。為了要體現出前述之定義「首次出現相關游玩特徵的得分幅度最顯著」的現象，我們對 y_1 的數值再進行開 c 次方根，分別以 $c = 2$ 、 $c = 3$ 與 $c = 4$ 得到 y_2 、 y_3 與 y_4 三種線性方程式，其中 y_2 的初期成長幅度為 $x_{22} - x_{21} = 0.1310$ 高於 y_1 同期的成長幅度，到了後期的成长幅度為 $x_{24} - x_{23} = 0.0847$ ，逐漸衰減同時已低於 y_1 同期的成長幅度，然而這樣的現象隨著常數 c 上升對於分數初期的成長幅度愈大、後期的成長幅度愈小。

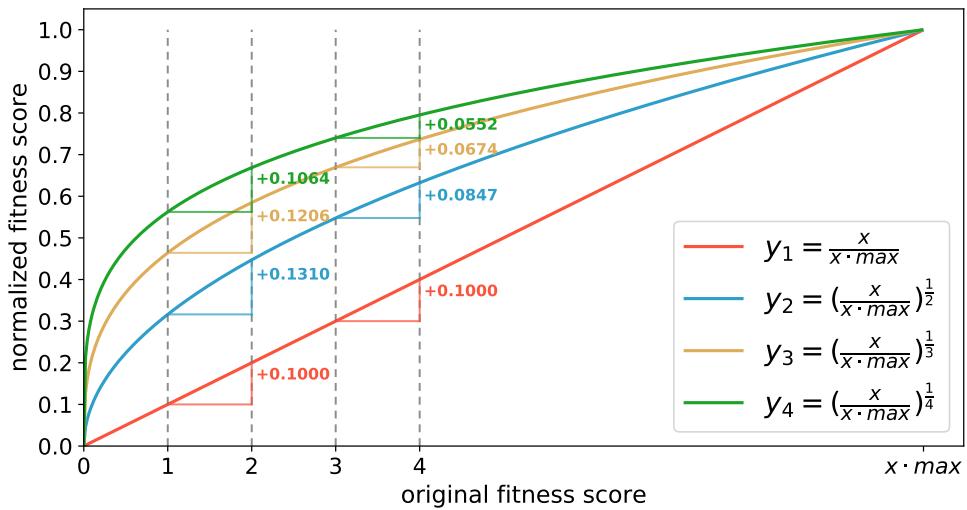


圖 3.29: 適應值進行標準化

3.3.6 套用物件演化機制的房間容器

綜合上述小節，不同房間容器類型會有對應的戰略配置。3.2.3 小節所介紹的房間容器外，在本小節追加了戰鬥通道的房間容器，與前章節提及之通道不同處，在戰鬥通道中玩家恐會遭遇敵方單位。戰鬥通道的格局是經過特殊設計的，敵方單位能夠依照不同房型格局進行對應的戰鬥策略，見圖 3.30。因此依照房間容器類型設置不同的適應性函數權重，逐一定義各自的適應性函數權重值。

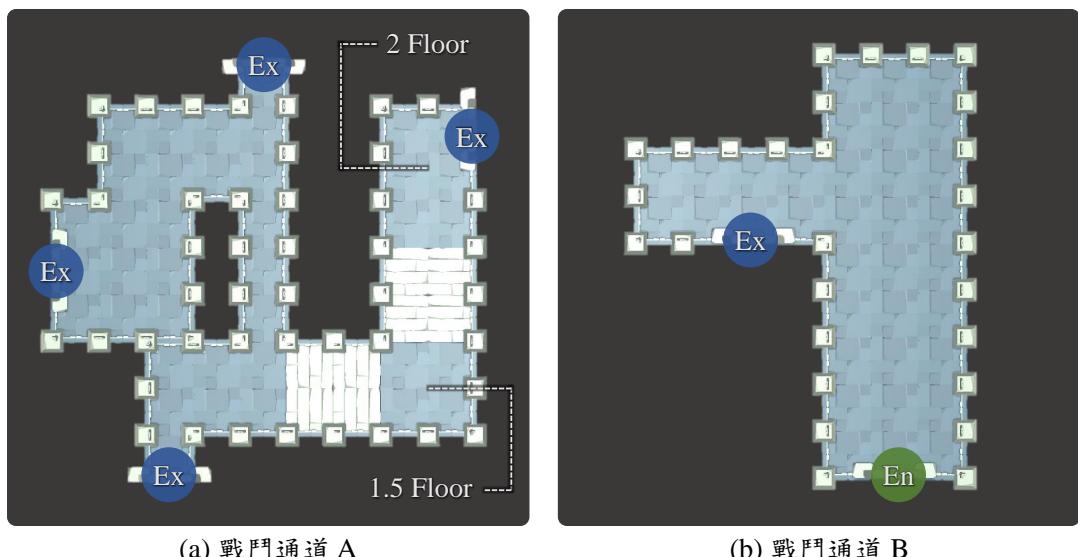


圖 3.30: 房間類型 - 戰鬥通道

寶藏房

寶藏房的端點識別物僅只有一個入口，便不存在空間動線，與空間動線相關的適應性函數（阻攔點、攔截點、支援點、至高點、陷阱點）不予採計。

- 守衛點權重: 1.00
- 遊戲物件數量限制: [3, 5]

Image is under construction

圖 3.31: 寶藏房的遊戲物件佈局演化示意



戰鬥通道（狹路驅逐）

- 此段僅為介紹，編輯中。
- 阻攔點權重: 1.00
 - 攔截點權重: 0.75
 - 陷阱點權重: 0.75
 - 遊戲物件數量限制: [2, 3]

Image is under construction

圖 3.32: 狹路驅逐情境的遊戲物件佈局演化示意

戰鬥通道（鎮守要道）

此段僅為介紹，編輯中。

- 阻攔點權重: 1.00
- 支援點權重: 0.75
- 巡邏點權重: 0.50
- 遊戲物件數量限制: [3, 5]

Image is under construction

圖 3.33: 鎮守要道情境的遊戲物件佈局演化示意



其餘房間容器

其餘未被定義的房型，像是入口、出口、一般通道... 等，因設計需求將不配置任何遊戲物件，即不參與相關的演化流程。

Image is under construction

圖 3.34: 將關卡中全部房間容器進行遊戲物件的佈局演化

第 4 章 實驗結果與分析

本章節將探討前述章節之方法是否符合研究目標，將進行以下實驗：第 4.3 小節中，針對所挑選的房間採用不同權重的適應性函數，會如何影響房間內的遊戲物件之配置結果。第 ?? 小節中，觀察染色體中的基因數量對於演化過程之影響。

4.1 實驗定義

我們使用 Invector 公司開發的 *Third Person Controller - Melee Combat Template* 套件（以下簡稱 3rdPC）進行遊戲物件的設置。3rdPC 提供遊戲開發者快速建構玩家控制角色 (Player Character)、非玩家控制角色 (Non-Player Character) 與其角色控制器或人工智能，並可製作任何類型的第三人稱動作冒險遊戲或角色扮演遊戲。第 4.1.1 小節中，說明 3rdPC 提供的玩家角色的操作使用說明；第 4.1.2 小節中，解釋實驗中的採用遊戲物件，其象徵意義與玩家角色互動方式；第 4.1.3 小節中，將定義屬實驗變因的相關參數設定，包含任務語法與基因演算法。

4.1.1 遊戲操作說明

玩家所控制的角色能夠移動、攻擊、防禦、跳躍與蹲下等動作。移動使用方向鍵，角色朝前後左右與其斜向組合鍵的方向移動，過程中按下 shift 按鍵將會切換至奔跑模式；攻擊使用滑鼠左鍵單擊，連續使用能夠進行三階段的攻擊，每次攻擊將消耗耐力值；防禦使用滑鼠右鍵單擊，能夠減少受到敵方攻擊的損傷；跳躍按鍵鍵盤 space 按鍵，角色能夠進行原地跳躍，在移動或奔跑的過程中亦可使用；蹲下按下鍵盤 c 按鍵，角色能夠緩慢的移動，適合觀察敵方戰況的同時進行移動。

4.1.2 遊戲物件說明

(編輯中)

敵人

敵人 (編輯中)。

寶箱

當玩家靠近並朝向寶箱時，按下鍵盤 E 按鍵便可以開啟寶箱，取得寶箱內容物。

陷阱

當玩家碰觸到陷阱時，會受到一定程度的損傷。

4.1.3 實驗參數設定



在任務語法階段時，我們將非合法... (編輯中)

地圖片段演化階段時，每一次世代的演化過程，有 80% 機率父母代間會進行兩點交配 (two-point crossover)；10% 機率衍生子代會進行突變，染色體個體中有 5% 至 20% 的基因數量會轉換成其它的物件種類。

4.2 資料收集

在資料收集階段中，將收集第 3.3 節中，房間容器進行基因演算法時，於實驗中各回合、世代與其個體（單一染色體）中的得分狀況。

4.3 演化結果與其品質

在第 3.3.6 小節中，展示了寶藏房與戰鬥通道（狹路驅逐、鎮守要道）三種空間的局部佈局演化結果，隨著房間容器搭配不同的適應性函數，便能夠生成出多

表 4-1: 原始資料之欄位示意

回合編號	世代編號	染色體編號	指標	得分	座標	類型	房間
1	1	1	Block	0	(12.0, 1.0, 0.0)	Empty	Room A
1	1	1	Intercept	0	(12.0, 1.0, 0.0)	Empty	Room A
1	1	1	Patrol	0	(12.0, 1.0, 0.0)	Empty	Room A
1	1	1	Guard	0	(12.0, 1.0, 0.0)	Empty	Room A
1	1	1	Support	0	(12.0, 1.0, 0.0)	Empty	Room A
1	1	1	Block	0	(24.0, 1.0, 12.0)	Empty	Room A
1	1	1	Intercept	0	(24.0, 1.0, 12.0)	Empty	Room A

樣性的遊戲物件佈局。在本小節中，將針對上述三種房間容器的演化結果做更進一步的分析。

寶藏房

編輯中。



戰鬥通道（狹路驅逐）

編輯中。

戰鬥通道（鎮守要道）

編輯中。

4.4 標準化結果之比較

在第 3.3.5 小節中，提出了方程式 3.8 的標準化效果會根據常數 c 的大小，影響適應值前期與後期的成長幅度。在本小節中，將改變常數 c ($c = 1$ 、 $c = 2$ 、 $c = 3$ 與 $c = 4$)，觀察應用在「單一指標型」與「複合指標型」房間容器的收斂情形。單一指標型意指僅採用一項適應性函數（不包含平衡適應性函數）；反之，複合指標型表示採用多項適應性函數。

單一指標型 - 寶藏房

編輯中。

複合指標型 - 戰鬥通道（狹路驅逐）

編輯中。

4.5 房型規模之比較

房型的大小較有可能直接影響和可行走瓦磚之數量。本階段的實驗中，我們提取第 4.2 節的資料，將空白、敵人兩種類型的數量關係繪製成熱圖進行觀察。這是因為各項適應性函數在設計時，多以「敵人」與其餘敵人、其它遊戲物件或玩家動線為考量參考，因而推估二者間勢必存在者某些關係。



第 5 章 結論與後續工作

5.1 貢獻與結論

我們提出將抽象化的動線作為評估的指標，驗證即使精確度降低的情形下，仍確保結果具有一定品質。(編輯中)。

5.2 限制與後續工作

在設計適應性函數的權重時，關卡設計師仍需要對於預期結果有一定的知識與掌握度，才能夠正確的產出預期的遊玩特徵。儘管增加了關卡設計師的工具使用門檻，若能熟悉本論文所提出的方法與工具相信對於遊戲開發上必有相當大的幫助。



參 考 文 獻

- [1] J. Chen, “Flow in games (and everything else),” *Communications of the ACM*, vol. 50, no. 4, pp. 31–34, 2007.
- [2] J. Dormans, “Adventures in level design: generating missions and spaces for action adventure games,” in *Proceedings of the 2010 workshop on procedural content generation in games*, p. 1, ACM, 2010.
- [3] J. Dormans, “Level design as model transformation: a strategy for automated content generation,” in *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, p. 2, ACM, 2011.
- [4] J. Dormans *et al.*, *Engineering emergence: applied theory for game design*. Creative Commons, 2012.
- [5] A. Liapis, G. N. Yannakakis, and J. Togelius, “Generating map sketches for strategy games,” in *European Conference on the Applications of Evolutionary Computation*, pp. 264–273, Springer, 2013.
- [6] A. Liapis, “Multi-segment evolution of dungeon game levels,” 2017.
- [7] D. Kazemi, “Spelunky Generator Lessons part 1: Generating the solution path.” <http://tiny-subversions.com/spelunkGen/>. Accessed: 2017-07-16.
- [8] D. Kazemi, “Spelunky Generator Lessons part 2: Generating the rooms.” <http://tiny-subversions.com/spelunkGen/>. Accessed: 2017-07-16.
- [9] A. Baldwin and J. Holmberg, “Mixed-initiative procedural generation of dungeons using game design patterns,” 2017.

附錄一：The binding of Isaac 的任務語法



附錄二：鎮守要道情境之演化釋例



附錄三：狹路驅逐情境之演化釋例



