



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

在本研究中，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，最終提供一完整的遊戲關卡生成解決方案。

我們將「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。透過將遊戲關卡的劃分為任務 (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同構轉換，並依照遊玩特徵定義基因演算法 (Genetic Algorithms) 的適應性函數，空間將透過基因演算法演化得出最適的遊戲物件佈局。

利用此關卡生成解決方案來設計關卡能有效減少開發時間、容易掌握遊戲各元件的配置外，同時讓玩家在進行遊戲時能夠遵循關卡設計師的劇情脈絡，亦能夠體驗到有意義且多樣化的遊戲關卡內容。

關鍵字：關卡生成、程序化生成、基因演算法

ABSTRACT

Under construction.



誌 謝

誌謝。



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	X
1 緒論	1
1.1 研究背景與動機	1
1.2 研究目的與研究問題	2
1.2.1 研究目的	2
1.2.2 研究問題	3
1.3 預計研究貢獻	3
1.4 本論文之章節結構	3
2 相關研究	4
2.1 關卡生成的方法	4
2.1.1 程序化生成任務內容	4
2.1.2 程序化遊戲物件擺放	5
2.2 基因演算法	6
3 研究方法	7
3.1 任務語法	7
3.1.1 建立任務符號表	8
3.1.2 建立任務規則	10
3.1.3 產生任務圖	14
3.2 空間建構	18
3.2.1 基礎結構	18
3.2.2 端點識別物	19
3.2.3 房間容器類型	19
3.2.4 從任務圖轉為遊戲空間	22
3.3 地圖片段	28

3.3.1	基因的儲存結構	28
3.3.2	演化之適應性函數	29
3.3.3	制衡取向的適應性函數	32
3.3.4	多項適應性函數合併	33
3.3.5	套用物件演化機制的房間容器	34
3.4	方法彙整與總結	37
4	實驗結果與分析	40
4.1	實驗定義	40
4.1.1	遊戲操作說明	40
4.1.2	遊戲物件說明	40
4.1.3	實驗參數設定與名詞解釋	41
4.2	實驗流程	42
4.2.1	透過任務語法建立任務圖	42
4.2.2	基於任務圖轉換為遊戲空間	42
4.2.3	資料收集	42
4.2.4	房間容器演化	42
4.3	房間容器佈局演化之結果	42
4.3.1	寶藏房	43
4.3.2	戰鬥通道（狹路驅逐）	51
4.3.3	戰鬥通道（鎮守要道）	58
4.3.4	完整關卡演化結果	64
5	結論與後續工作	65
5.1	貢獻與結論	65
5.2	限制與未來研究方向	65
參考文獻		66
授權書		67

圖 目 錄

圖 1.1 Rogue 的遊戲畫面	2
圖 2.1 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	7
圖 3.2 Dungeon Generator 工具	8
圖 3.3 任務規則範例	10
圖 3.4 建立主線任務的預期任務圖與空間預覽	11
圖 3.5 於 Dungeon Generator 工具中，若規則為非法狀態將不予以生效	14
圖 3.6 任務語法之改寫系統流程	15
圖 3.7 任務圖的生成過程	17
圖 3.8 根據不同亂數種子所輸出的任務圖	17
圖 3.9 影響遊戲性的遊戲物件，由左至右分別為敵方、寶箱與陷阱	18
圖 3.10 房間容器的體素建置方式，與五種裝飾物	19
圖 3.11 端點識別物的實際使用與串接情形	20
圖 3.12 起點、終點與魔王房的房間容器結構	20
圖 3.13 通道房間的房間容器結構與其同構示意	21
圖 3.14 石牆、秘密房、鎖、商店與寶藏房的房間容器結構	22
圖 3.15 死路的房間容器結構	22
圖 3.16 建造方式的疊代生成過程	25
圖 3.17 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	26
圖 3.18 經過剩餘空間替換後的完整遊戲空間	27
圖 3.19 房間容器採取基因演算法之演化流程	28
圖 3.20 房間容器以遊戲物件的佈局方式作為染色體	29
圖 3.21 動線於房間容器的計算方式，圖中數值為行經次數	30
圖 3.22 守衛點指標的遊戲體現情形	30
圖 3.23 阻擋點指標的遊戲體現情形	31
圖 3.24 巡邏點指標的遊戲體現情形	32
圖 3.25 適應值進行標準化	34
圖 3.26 擴充房間容器，戰鬥通道的房間容器結構	35

圖 3.27 寶藏房的遊戲物件佈局演化示意	35
圖 3.28 狹路驅逐情境 II 的遊戲物件佈局演化示意	36
圖 3.29 鎮守要道情境的遊戲物件佈局演化示意	37
圖 3.30 將關卡中全部房間容器進行遊戲物件的佈局演化	38
圖 3.31 本論文提出之方法，完整框架示意圖	39
圖 4.1 實驗 Treasure-1 的演化結果及各演化耗時	45
圖 4.2 實驗 Treasure-2 的演化結果及各演化耗時	47
圖 4.3 實驗 Treasure-3 的演化結果及各演化耗時	49
圖 4.4 實驗 Narrow-1 的演化結果及各演化耗時	52
圖 4.5 實驗 Narrow-2 的演化結果及各演化耗時	54
圖 4.6 實驗 Narrow-3 的演化結果及各演化耗時	56
圖 4.7 實驗 Trunk-1 的演化結果及各演化耗時	59
圖 4.8 實驗 Trunk-2 的演化結果及各演化耗時	61
圖 4.9 實驗 Trunk-3 的演化結果及各演化耗時	63



表 目 錄

表 1.1 Joris Dormans 與 Antonios Liapis 提出之方法，進行綜觀比較	2
表 3.1 系統預設的任務符號	8
表 3.2 利用遊玩特徵建立任務符號表之範例	9
表 3.3 線性任務規則範例，建立主線任務	11
表 3.4 非線性任務規則範例，設置特殊房	12
表 3.5 非線性任務規則範例，建立支線任務	12
表 3.6 非線性任務規則範例，鎖的提前出現與房間攤平	13
表 3.7 非法的任務規則定義	13
表 3.8 建造方式表，任務終端節點與房間容器對應之範例	24
表 3.9 空間替換表，出口端點與房間容器對應之範例	26
表 4.1 實驗使用之名詞釋義	41
表 4.2 演化適應值資料節錄示意	43
表 4.3 演化座標資料節錄示意	43
表 4.4 實驗 Treasure-1 之基因演算法參數配置	45
表 4.5 實驗 Treasure-1 - 共 10 回合的最佳個體之標準化加權適應值	46
表 4.6 實驗 Treasure-2 之基因演算法參數配置	47
表 4.7 實驗 Treasure-2 - 共 10 回合的最佳個體之標準化加權適應值	48
表 4.8 實驗 Treasure-3 之基因演算法參數配置	49
表 4.9 實驗 Treasure-3 - 共 10 回合的最佳個體之標準化加權適應值	50
表 4.10 實驗 Narrow-1 之基因演算法參數配置	52
表 4.11 實驗 Narrow-1 - 共 10 回合的最佳個體之標準化加權適應值	53
表 4.12 實驗 Narrow-2 之基因演算法參數配置	54
表 4.13 實驗 Narrow-2 - 共 10 回合的最佳個體之標準化加權適應值	55
表 4.14 實驗 Narrow-3 之基因演算法參數配置	56
表 4.15 實驗 Narrow-3 - 共 10 回合的最佳個體之標準化加權適應值	57
表 4.16 戰鬥通道（鎮守要道）演化結果之平均適應值與總體標準差 σ	58
表 4.17 實驗 Trunk-1 之基因演算法參數配置	59
表 4.18 實驗 Trunk-1 - 共 10 回合的最佳個體之標準化加權適應值	60

表 4.19 實驗 Trunk-2 之基因演算法參數配置	61
表 4.20 實驗 Trunk-2 - 共 10 回合的最佳個體之標準化加權適應值	62
表 4.21 實驗 Trunk-3 之基因演算法參數配置	63
表 4.22 實驗 Trunk-3 - 共 10 回合的最佳個體之標準化加權適應值	64



演 算 法 目 錄

演算法 1 RewriteSystem1 - 改寫系統（任務語法）	16
演算法 2 Part of RewriteSystem1 - 搜尋匹配規則	16
演算法 3 RewriteSystem2 - 改寫系統（任務轉換空間）	23



第 1 章 緒論

程序化內容生成 (Procedural Content Generation，以下簡稱 PCG) 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。在本研究中，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



在電腦圖學領域中，經常利用到 PCG 的技術來解決貼圖 (texture) 的生成、3D 模型的生成等。只需要運行的程式函數與基本的素材資源，便能夠藉由電腦快速運算出大量、豐富的內容，而這樣的特性相當適合被應用在電子遊戲領域中。PCG 遊戲開發技術已實行多年，採用相關技術開發的經典遊戲中，最早可追溯至由 Michael Toy 和 Glenn Wichman 於 1980 年左右開發的經典電子遊戲《Rogue》，圖 1.1。近年來，伴隨著遊戲產業的競爭下，成本的考量更加地重視，如涉及地圖場景、關卡設計的部分相當耗費人力成本。而 PCG 的遊戲開發技術，便是為了解決降低開發成本的同時保持遊戲性，甚至跳脫出人類的思考框架，進而創造出更豐富多樣的遊戲內容，因此這樣的技術又再度成為熱門焦點。

眾多開發團隊導入 PCG 技術至自家產品中，希冀藉此提升遊戲開發速度與其成本效益，不過目前的程序化生成技術還沒有辦法完全的取代人力。生成可遊玩的遊戲關卡是最低需求，僅以可遊玩為目標的話，現今已存在大量的程式與產品能夠達成此項目標。倘若需要兼顧內容品質與數量，那麼符合玩家遊戲體驗期待的 PCG 方法就顯得相當稀少，甚至只能針對特定的遊戲主題或產品。且隨機生成的內容是否能受玩家青睞與喜愛，仍是眾團隊需要研究的課題。

在本研究中，我們不涉及遊戲有趣程度的議題，緣由是受到遊戲本身的劇本



圖 1.1: Rogue 的遊戲畫面

主題、美術設計與關卡企劃等不同面向，因此我們把研究目標定位在如何藉由 PCG 技術，生成符合遊戲內容需求的地圖，藉由宏觀的抽象遊玩特徵，將其利用自動生成的方式實現出，並確保遊玩特徵的概念確實地落實在其生成結果當中。

1.2 研究目的與研究問題

由 Joris Dormans 提出的自動關卡設計方法中，藉由任務語法精準控制玩家遊玩的進程，而遊戲物件的佈局由空間語法掌控，卻無法控制其品質優劣。由 Antonios Liapis 提出的戰略型遊戲抽象化地圖生成方法中，對於遊戲物件的佈局提供了評定品質的量測方式，表 1.1 對上述二者進行了簡易比較。

表 1.1: Joris Dormans 與 Antonios Liapis 提出之方法，進行綜觀比較

方法框架	任務進程	內容評估
Mission/Space 框架 (Joris Dormans)	精準控制	無法評估優劣
Sentient Sketchbook (Antonios Liapis)	未提及	提供優劣評估

1.2.1 研究目的

我們將參考上一段落所提及的兩種方法框架，針對遊戲關卡的意義性與關卡品質進行研究。研究目的參照下列所述：

1. 設計並提出能夠生成帶有意義的 3D 動作遊戲的關卡自動生成演算法。
2. 以基因演算法輔助該自動生成演算法，進行自我品質評估與改良。

1.2.2 研究問題

根據上述之研究目的，本研究提出以下研究問題：

1. 存在多種不同面向的適應性函數，要如何進行有效的數值標準化？
2. 人為調整相關基因演算法演化參數時，對於關卡生成會有什麼影響？

1.3 預計研究貢獻

我們預期本論文提及之方法進行整合與改良，最終提供關卡設計師一完整的遊戲關卡生成解決方案。並提出高度語意化的遊玩特徵指標，讓關卡設計師不須透過撰寫程式碼或設計數學模型，僅需調整各項遊玩指標的權重值便能生成出對應的複合型遊玩特徵。

1.4 本論文之章節結構

本論文第 2 章為相關研究，涵蓋了程序化生成技術應用於關卡生成之方法以及基因演算法。第 3 章為研究方法分為三節：其一，將巨觀的遊玩流程透過任務語法建立任務圖；其二，建構房間容器，並透過改寫系統將任務圖轉換為遊戲空間；其三，將關卡的各個房間容器，使用語意化的遊玩指標搭配基因演算法生成合適的遊戲物件佈局。第 4 章為實驗結果與分析，將呈現前述方法的系列實作結果，並且針對程序化生成的內容進行結果分析。第 5 章為結論與後續工作，為本論文提出之方法及實作內容之結果進行總結，以及未來研究目標與其願景。

第 2 章 相關研究

2.1 關卡生成的方法

本節介紹數種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 2.1.1 小節的程序化生成任務內容與 2.1.2 小節的程序化遊戲物件擺放。

2.1.1 程序化生成任務內容

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 [1] [2] [3]；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 [1] [3]，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)，二者能夠獨立分別生成出任務圖與空間圖，倘若將改寫系統套用在任務圖上，便使其產生出與任務圖同構的遊戲空間。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。

2.1.2 程序化遊戲物件擺放

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 [4]。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 (Flood Fill Algorithm)，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 [5]，見圖 2.1。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。

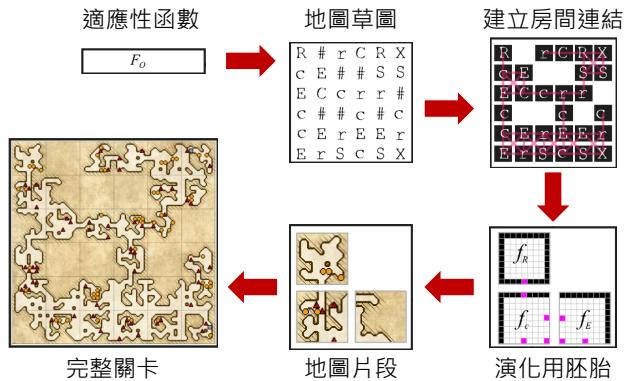


圖 2.1: Antonios Liaps 提出的兩階段式關卡演化



2.2 基因演算法

基因演算法由 John Holland [6] 首次提出。經學界多年來的許多研究，基因演算法已被證實為一有效最佳化搜尋方法。根據生物學家 Charles Darwin 進化論的演化與適者生存的觀點，藉由數學科學模擬出「物競天擇，適者生存」的生物演化法則，其演化與淘汰的概念便是效仿自然界生物的遺傳與進化，物種將根據生存環境調整自身物種的基因組合，使其後代適應該自然環境。

John Holland 藉由電腦模擬出物種的生存模式，每一物種作為族群 (population)，群組由多個不同個體 (individual) 所組成，每一個體擁有自己的染色體 (chromosome)，染色體由基因 (gene) 的序列所構成。基因演算法的基礎架構由適應系統 (Adaptive System) 與進化系統 (Evolutionary System) 所構成 [6]：適應系統中，將留下較優良的個體，淘汰較差的個體，評定方法藉由適應性函數 (fitness function) 估算其染色體的適應值 (fitness)，較高的適應值總和便列為優良個體；進化系統中，透過數據化基因序列的結構，將該序列視為二進制的編碼，隨著個體相互進行交配 (crossover)、子代發生突變 (mutation)，都會讓基因序列變得多樣化。結合適應系統與進化系統，物種便隨著時間流逝形成多個世代 (generation)。

第3章 研究方法

本論文中，我們仍保持 Joris Dormans [1] [3] 與 Antonios Liapis [4] 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 3.1 a，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 (mission graph)。圖 3.1 b-c，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 (volume)，並得到尚未包含遊戲物件的遊戲地圖。圖 3.1 d-e，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 (fitness functions)，藉由基因演算法演化房間容器的流程，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。

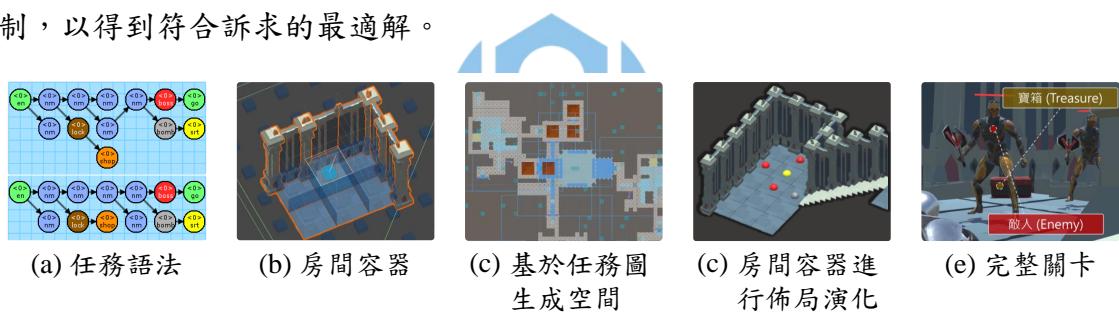


圖 3.1: 本論文提出系統之流程

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，見圖 3.2。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 [7] 作為範例，解析其遊玩特徵並作為任務語法的節點。表 3.1 為系統預設任務符號表。

表 3.1: 系統預設的任務符號

代號	名稱	符號	說明
en (entrance)	起點	en	玩家於遊戲關卡開始時的起始位置。
go (goal)	終點	go	玩家抵達此位置時遊戲將結束。
? (any)	任一節點	any	此節點能夠被判別為任何節點，定義任務規則時有著特殊用途。於 3.1.2 小節中，對其使用方式將進行闡述。

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，表 3.2 列舉利用遊玩特徵所建立的任務符號表，藉此歸納出「起

點 (entrance )、終點 (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 )」，其中起點與終點為系統預設之終端節點。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。

表 3.2: 利用遊玩特徵建立任務符號表之範例

代號	名稱	符號	說明	類型
S (Start)	起始節點		任務圖進入改寫階段時，系統預設的根節點。	起始節點
boss (boss)	魔王房		房間中存在著較強大的怪物。	主線任務
NM (Normal)	一般房		一般房間，藉由其它規則替換成其它房間或事件。	
nm (normal)	一般房		遊戲過程中經過的通道，經常出現在遊戲關卡中。	特殊事件
bomb (bomb)	使用炸彈		必須使用炸彈將石牆給破壞，破壞石牆後便可通過。	
srt (secret)	秘密房		能夠取得稀有道具鑰匙的房間。	
SP (Special)	特殊房		特殊房間，藉由其它規則替換成不同的稀有房間。	
lock (lock)	鎖		必須使用鑰匙才能夠通過該空間。	
shop (shop)	商店		能夠觸發特殊事件，玩家能夠在此購買道具。	
ts (treasure)	寶藏房		藏有稀有道具的房間，當中會有守衛寶箱的敵人。	

3.1.2 建立任務規則

圖 3.3，任務規則分為規則左側及規則右側，二者皆為圖形語法 (Graph Grammars) 所構成，圖形與則由終端節點、非終端節點與連接線 (connection) 所構成的有向圖。規則左側為被取代方、規則右側為取代方，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，可參照第 3.1.3 小節之說明。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，多為了因應部分遊戲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。

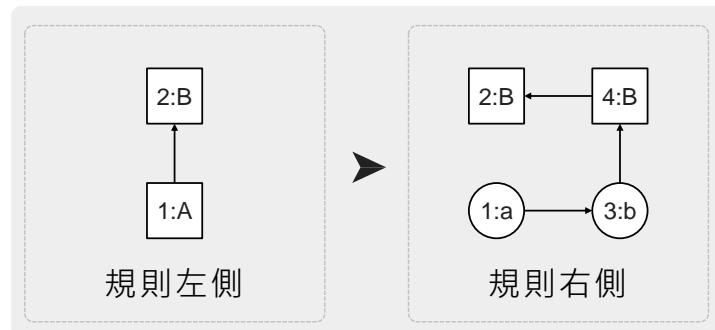


圖 3.3: 任務規則範例

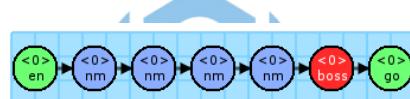
線性任務規則

玩家在進行遊戲時，會有一條的主要任務流程、劇情安排。表 3.3 設計了三道任務規則，分別為主線任務、配置魔王房與探索通道。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 en) 與終點 (goal go) 所構成的線性任務，其中兩節點之間由數個非終端節點的一般房 (Normal NM) 相連，而非終端節點的一般房會倚賴其它規則進行後續替換。配置魔王房規則中，規則左側中定義了必須為非終端節點的一般房 (Normal NM) 與終點 (goal go) 相連，並改寫該一般房為魔王房 (boss boss)。探索通道規則中，會將任務圖中的剩餘非終端節點的一般房 (Normal NM) 替換為終端節點的一般房 (normal nm)。

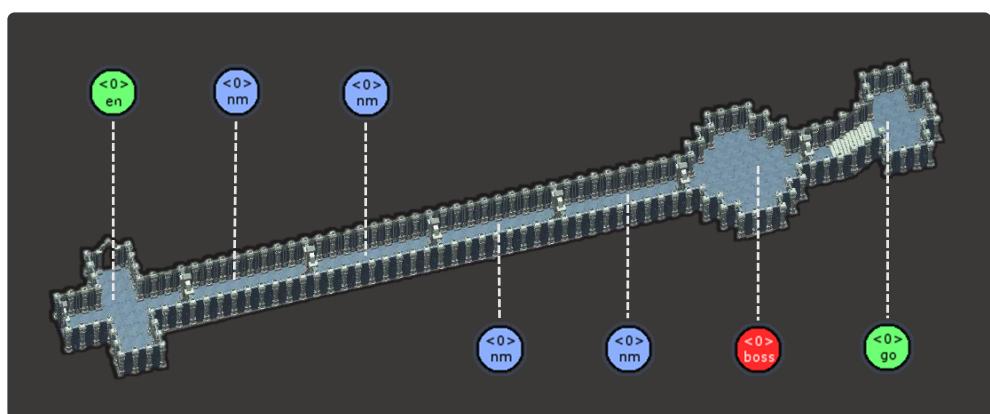
表 3.3: 線性任務規則範例，建立主線任務

代號	名稱與任務規則	
Main Path	主線任務	
Boss Room	配置魔王房	
Exploration	探索通道	

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 3.4 的空間。



(a) 採用的任務圖



(b) 能夠經由 (a) 所生成的遊戲空間

圖 3.4: 建立主線任務的預期任務圖與空間預覽

非線性任務規則

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，表 3.4 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) 與任一節點相連的圖形語法（任一節點是系統保留

的節點，再圖形語法中能夠表示任何的節點)，會被轉換為規則右側的非線性任務圖形語法，非線性的定義是當一個節點與複數個節點連接，構成分支結構，當中的秘密房 (secret^{srt}) 能夠獲取鑰匙，並有需要由炸彈才能夠突破的牆壁 (bomb^{bomb}) 與其它區域阻隔，此外延伸出特殊房 (Special^{SP}) 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^{SP}) 能夠被改寫為鎖 (lock^{lock}) 與商店 (shop^{shop}) 或寶藏房 (treasure^{ts}) 相連的遊玩特徵。

表 3.4: 非線性任務規則範例，設置特殊房

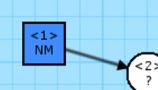
代號	名稱與任務規則	
Set Secret	配置特殊房	 
Shop	商店	 
Treasure	寶藏房	 

表 3.5，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^{NM}) 便能再經過其它的規則，取代成不同的遊玩特徵。

表 3.5: 非線性任務規則範例，建立支線任務

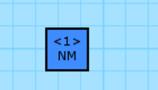
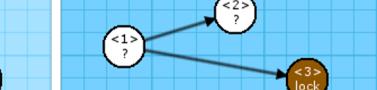
代號	名稱與任務規則	
More Branch	建立支線	 

表 3.6，為使玩家進行遊戲能夠提前遇見鎖 (lock^{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。

表 3.6: 非線性任務規則範例，鎖的提前出現與房間攤平

代號	名稱與任務規則	
Forward Lock	鎖向前	 
Flat Rooms	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，見圖 3.5。例如：改寫系統陷入無限循環或任務圖破碎化。在表 3.7 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。

表 3.7: 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節點，便無法正確定位出任務起點。

接續下頁面

表 3.7 – 接續前頁面

非法規則之標籤	標籤的狀況描述
OverflowedAnyNode	當規則右側使用 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，規則左側使用 Any 節點將不在此限。

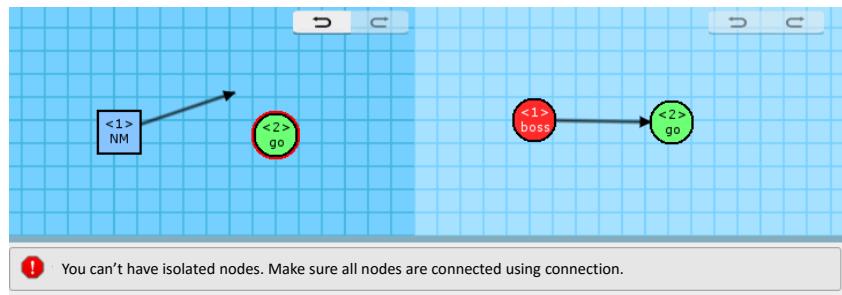


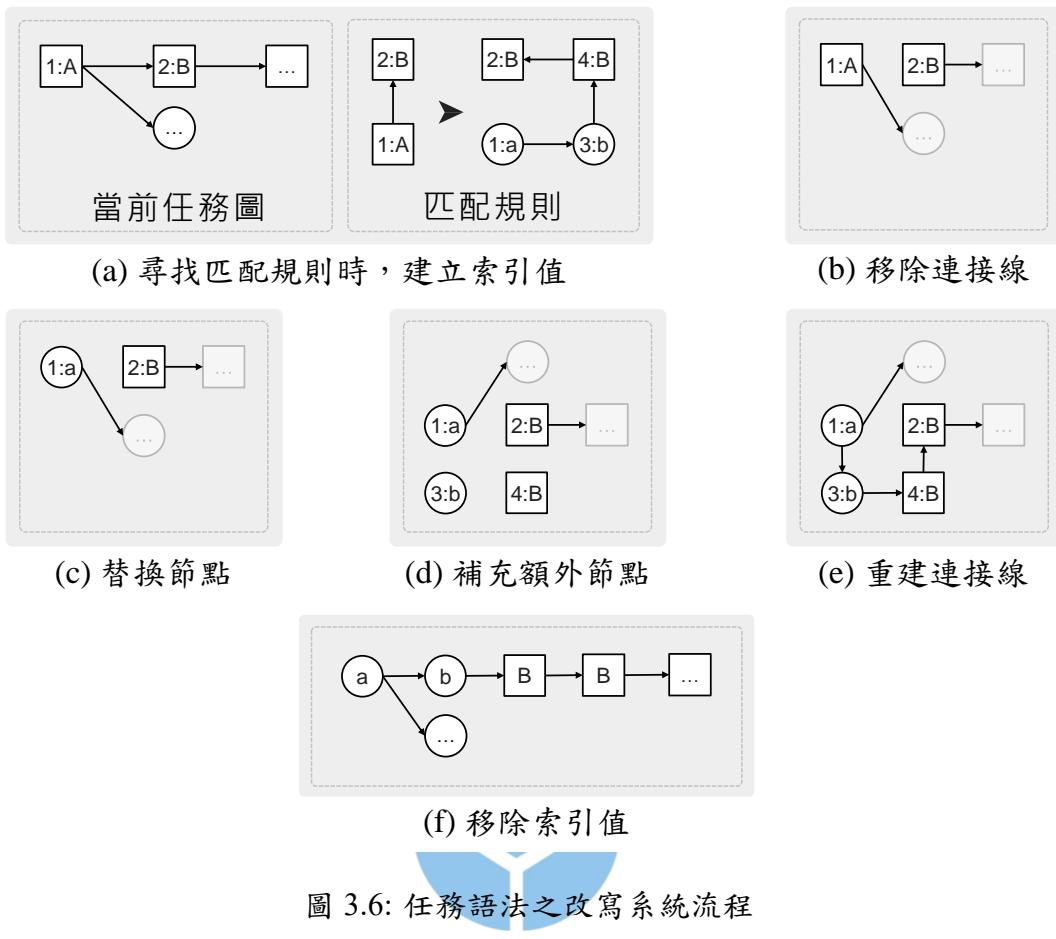
圖 3.5: 於 Dungeon Generator 工具中，若規則為非法狀態將不予以生效



3.1.3 產生任務圖

上一小節中定義了任務規則，本小節將利用任務規則進行生成任務圖的步驟。我們參考 Joris Dormans [1] 提出的方法並改良之，歸納出演算法 1 - 改寫系統（任務語法）與演算法 2 - 搜尋匹配規則。圖 3.6，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。

圖 3.6 a，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) [8] 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時，有關連的節點會標記與匹配規則一致的索引值。圖 3.6 b，將任務圖含有索引值的節點，其相連的連接線移除。圖 3.6 c，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。圖 3.6 d，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。圖 3.6 e，依照匹配規則右側的連接線，以相同方式放入任務圖中。最後一步驟圖 3.6 f，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。

圖 3.7 根據任務規則表（表 3.3- 3.6）生成任務圖，並使用亂數種子 894730 作舉例，每次的疊代生成依照演算法 1 將當前任務圖的所有節點遍歷 (traversal)。根據前述章節定義，當圖 3.7 d 時已不存在非終端節點，任務圖已趨穩定，但因為鎖向前規則與房間攤平規則，其規則左側皆為終端節點的情形下，儘管任務圖已趨穩定，只要任務圖中仍有同構子圖存在，改寫系統便能持續運行下去。在 *Dungeon Generator* 中，亂數種子會同時決定一隨機次數，讓已趨穩定的任務圖仍能夠在有限次數下持續進行疊代，直至無任何規則可以被採用，見圖 3.7 g。在圖 3.8 中展示了根據不同亂數種子，能夠得到豐富結果的任務圖。

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 and root doesn't be explored then
2:   matchedRule = FindMatchs(root)                                ▷ 演算法 2 之方法
3:   移除與 matchedRule 相符子圖節點間的連接線
4:   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 ∈ children of root do:
10:   RewriteSystem1(child, rules, seed)                               ▷ 遞迴式，跳往 Line 1
11: end for
12: return root
```



Algorithm 2 Part of RewriteSystem1 - 搜尋匹配規則**Input:**

root is current *pointer* in the mission graph.

Output:

```
1: matchedRules is an empty array of rule
2: graph = TransformToGraph(root)                                     ▷ 以該節點作為新圖之根
3: for all rule ∈ GetAllRules() do:                                         ▷ 參照任務規則表，表 3.3- 3.6
4:   if graph is isomorphic with left of rule then                      ▷ 若當前圖與該規則左側同構
5:     matchedRules add rule                                              ▷ 將該規則加入候選名單
6:   end if
7: end for
8: selectedRule = RouletteWheel(matchedRules)  ▷ 採輪盤法，return rule or null
9: return selectedRule
```

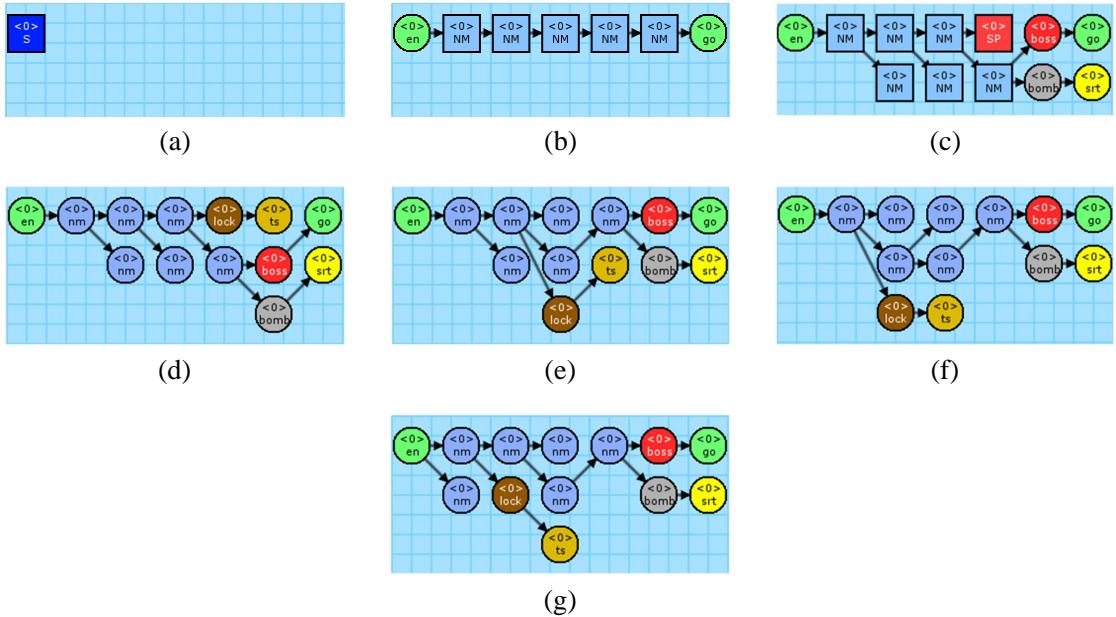


圖 3.7: 任務圖的生成過程

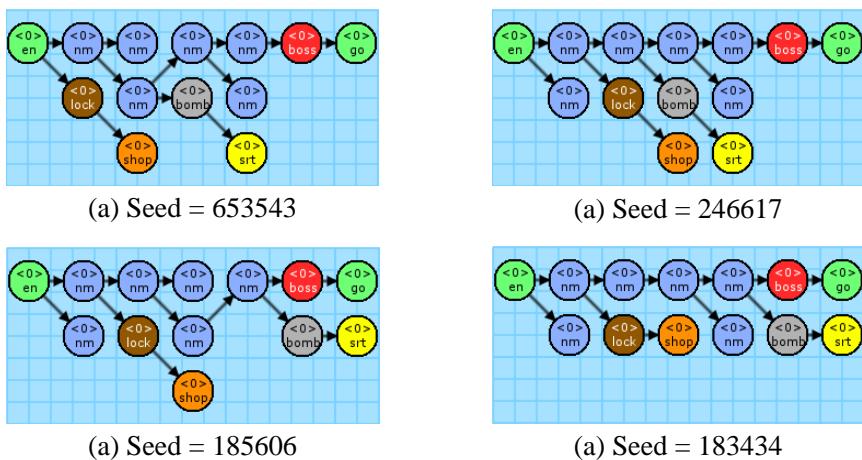


圖 3.8: 根據不同亂數種子所輸出的任務圖

3.2 空間建構

Joris Dormans 於文獻中提到為二維空間的範例 [1] [3]，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 3.9。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 3.9: 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。在一個關卡 (level) 中包含數個房間容器 (volumes)，每一房間容器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。圖 3.10 a-c 可見，進行房間容器編輯模式下，選定一裝飾物後需透過提示色框進行位置的確認與擺放，水藍色立方體為當前體素，紅色方框為當前方位。不同的裝飾物會依照其特性決定能夠放置的方位，如地面則放置於中心點，圖 3.10 a；牆壁、階梯或門放置於正四方，圖 3.10 b；牆壁柱放置於斜四方，圖 3.10 c。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 3.10 d-g 中，「地面、階梯、牆壁、牆壁柱、門」共五種裝飾物進行房間容器的體素建置工作。

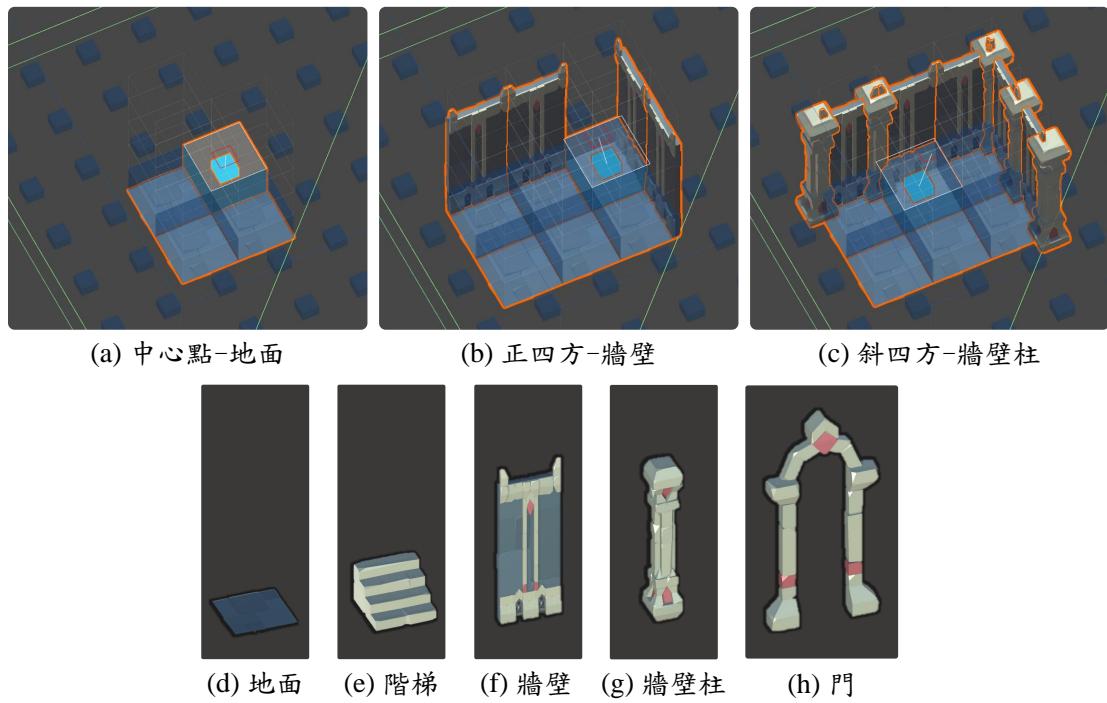


圖 3.10: 房間容器的體素建置方式，與五種裝飾物



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 [3]，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 3.11 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。

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，依照節點性質設計房型，區分出數項不同房型的空間規劃。

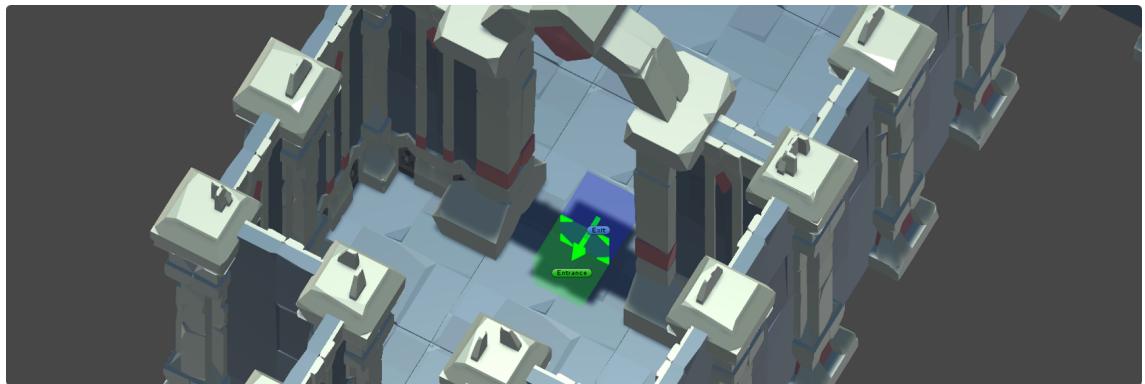


圖 3.11: 端點識別物的實際使用與串接情形

起點、終點、魔王房

圖 3.12 a 為起點擁有一個入口與三個出口，為玩家重生的起始房間；圖 3.12 b 為終點擁有一個入口，為遊戲關卡結束的房間；圖 3.12 c 為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。

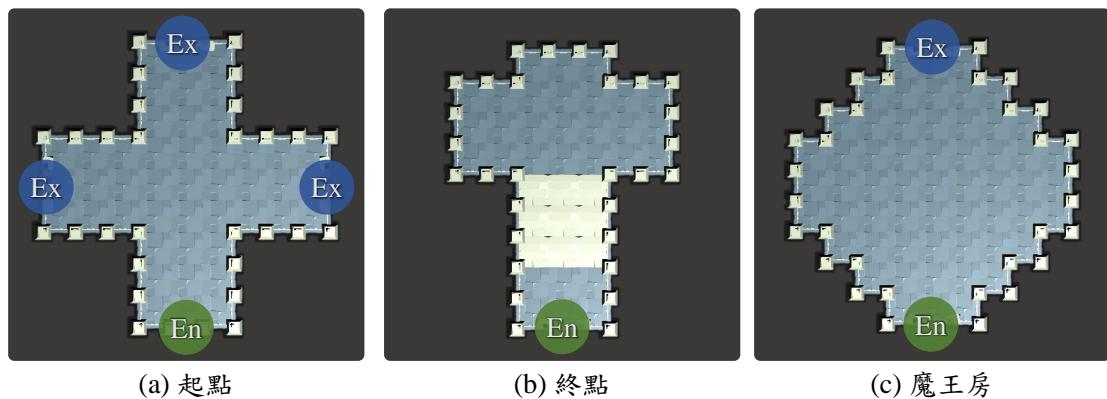


圖 3.12: 起點、終點與魔王房的房間容器結構

通道

圖 3.13 中，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 3.13 a 為 I 型通道擁有二個出口，有二種同構結構；圖 3.13 b 為 L 型通道擁有二個出口，有四種同構結構；圖 3.13 c 為 T 型通道擁有三個出口，有四種同構結構；圖 3.13 d 為 + 型通道擁有四個出口。

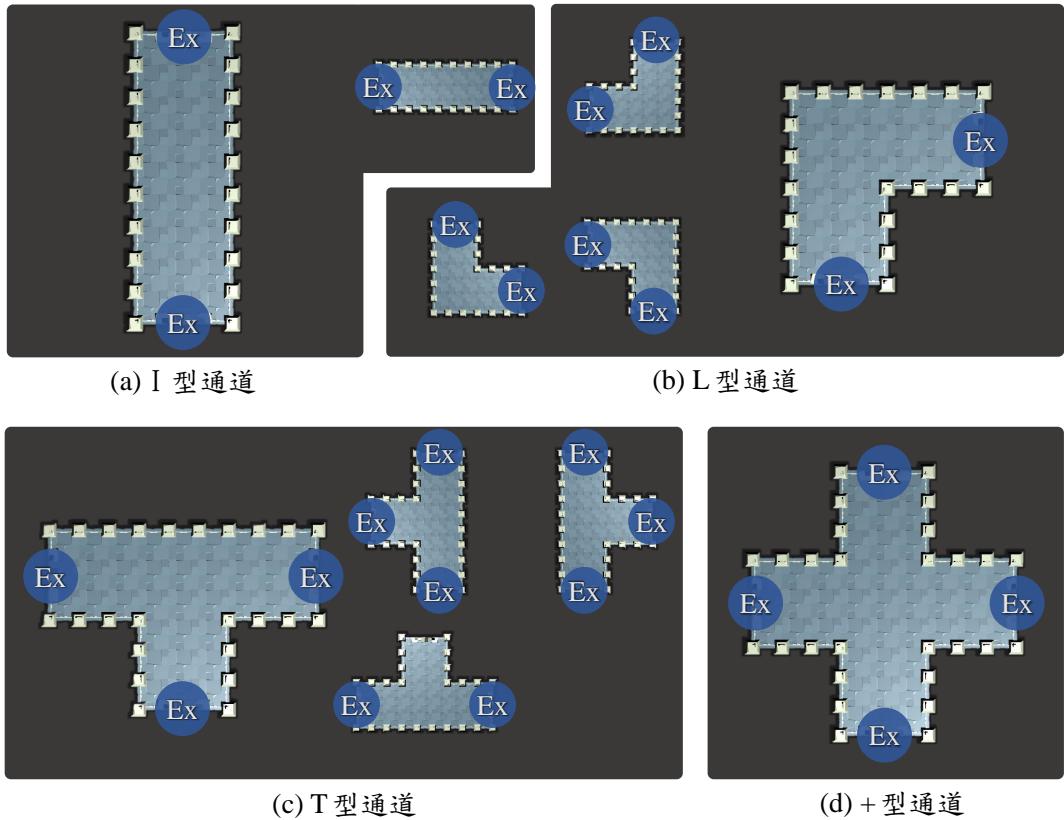


圖 3.13: 通道房間的房間容器結構與其同構示意

石牆、秘密房、鎖、商店、寶藏房

圖 3.14 a 為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；圖 3.14 b 為秘密房擁有一個入口，玩家能在該空間取得鑰匙識別物；圖 3.14 c 為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；圖 3.14 d-e 為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。

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，見圖 3.15。

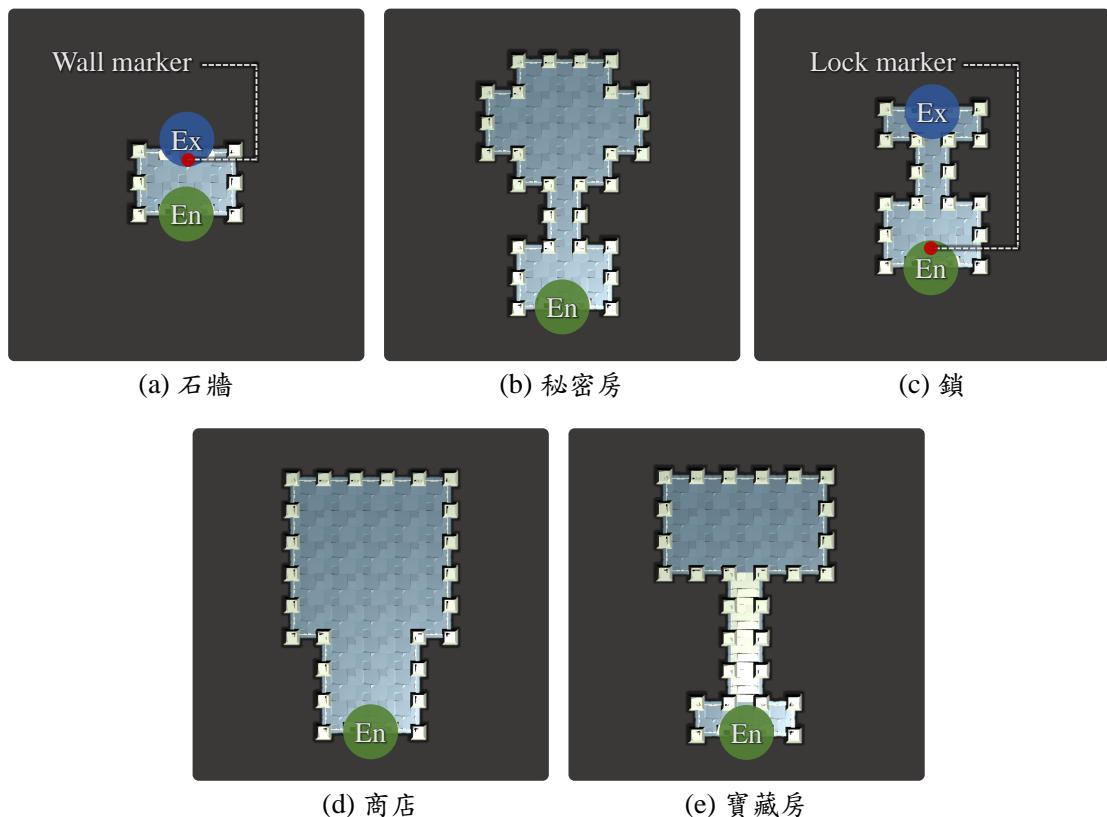


圖 3.14: 石牆、秘密房、鎖、商店與寶藏房的房間容器結構



圖 3.15: 死路的房間容器結構

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 [1]。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」（表 3.8）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」（表 3.9）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 3 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，經過的源節點便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，接著透過空間替換表，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，相關演示可參考第 3.2.4、3.2.4 二小節，將會逐一介紹演算法 3 的實際應用流程。

Algorithm 3 RewriteSystem2 - 改寫系統（任務轉換空間）

Input:

graph is *MissionGraph*.

seed is the random seed.

Output:

```

1: for all node  $\in$  graph do:                                ▷ Instrcution , 第 3.2.4 小節
2:   volume = GetVolumeFromInstructionTable(node, seed)      ▷ 參照表 3.8
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  $\in$  ConnectionsInGraph do:          ▷ Replacement , 第 3.2.4 小節
8:   volume = GetVolumeFromReplacementTable(marker, seed)    ▷ 參照表 3.9
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

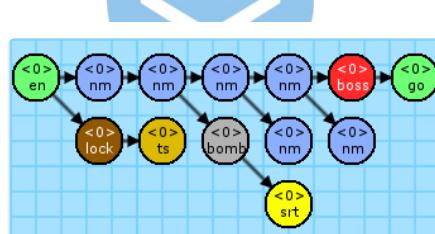
為從已生成完畢的任務圖進而衍生出遊戲空間，在創建的房間容器會對應一任務符號表當中的終端符 (terminal symbols)，稱作為建造方式 (building instruction)。本小節會利用第 3.1.1 小節的任務符號表，過濾出終端節點並逐一對應至第 3.2.3 小節的房間容器，見表 3.8。每一個終端節點都必須對應一個以上的房間容器，房間容器不允許被重複使用在不同的終端節點上。

表 3.8: 建造方式表，任務終端節點與房間容器對應之範例

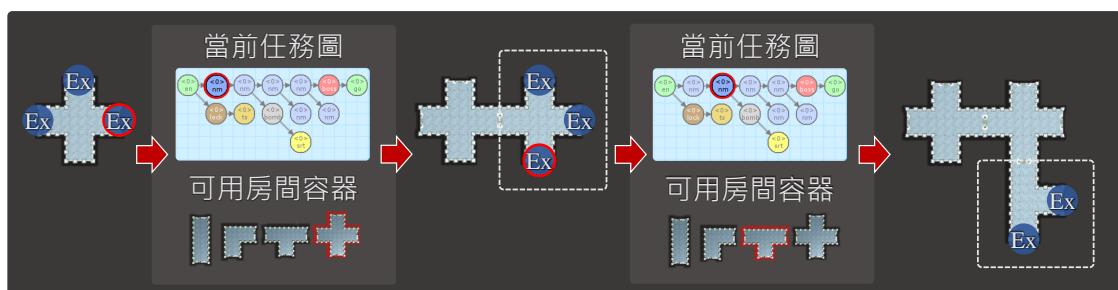
節點	對應之房間容器	節點	對應之房間容器
en		go	
boss		nm	
nm		nm	
nm		bomb	
srt		lock	
shop		ts	

圖 3.16 a 為使用第 3.1.2 小節的任務規則表（表 3.3- 3.6）所生成的任務圖，該階段採用的亂數種子為 0。本階段中，依照演算法 3 - 改寫系統（任務轉換空間），並且結合表 3.8 的建造方式表，將任務圖轉換為遊戲空間。圖 3.16 中，第一個節點起點 (entrance en) 直接對應到「起點房間容器」，而起點房間容器中擁有三個出口端點，系統將會亂數隨機一個出口端點進行後續房間生成，根據深度優先搜尋法下一個遍歷的節點為第二行的一般房 (normal nm)，接著從建造方式表中過濾出能夠使用的四種「通道房間容器」，由系統亂數決定本次採用的房間容器後，將預計新增的房間容器之起點與前一步驟決定的出口端點進行房間容器連接，下一回合將從新增的房間容器重複前述之步驟。依照上述之步驟，改寫系統（任務轉換空間）完整遍歷所有的任務節點後，便能夠完成任務轉換空間的動作，結果可參考圖 3.17。

進行疊代的過程時，以下幾種情形，疊代生成將會失敗：當任務圖節點的子節點數量大於對應房間容器的出口數量；當新生長的房間容器於其它房間容器發生碰撞。因此，若有以上違例情況，關卡設計師必須修正其關卡設計，如調整房間容器之格局或更換一筆亂數種子，並重新執行前述步驟。



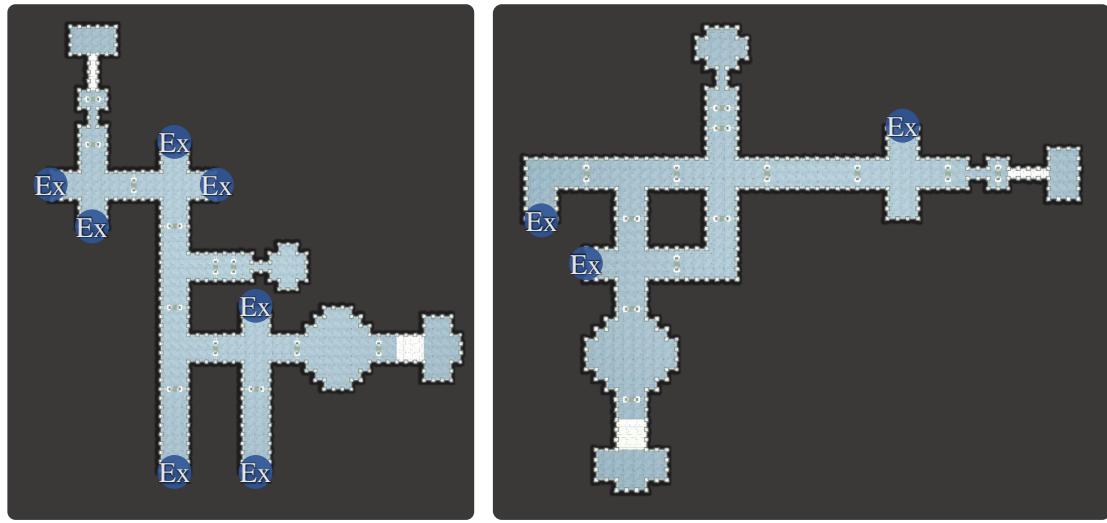
(a) Seed = 0



(b) Seed = 1，任務轉換空間的疊代過程

圖 3.16: 建造方式的疊代生成過程

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 3.17。



(a) Seed = 1, 生成結果

(b) Seed = 4, 生成結果

圖 3.17: 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

剩餘空間替換



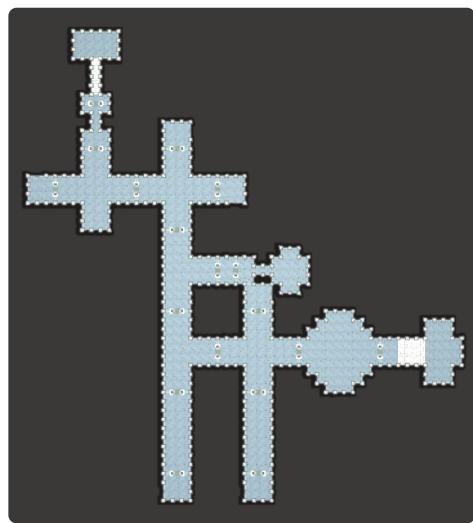
一個端點識別物能夠對應複數個房間容器，房間容器能夠重複被不同的端點識別物所對應。第 3.2.2 小節，介紹端點識別物時，曾提及到遊戲設計師能夠自由擴充出口識別物的種類。

在此為簡化範例，表 3.9 採用預設之單一出口識別物，而這項出口識別物將對應圖 3.15 的死路房間容器，左側為端點識別物之種類，右側為所對應的房間容器。

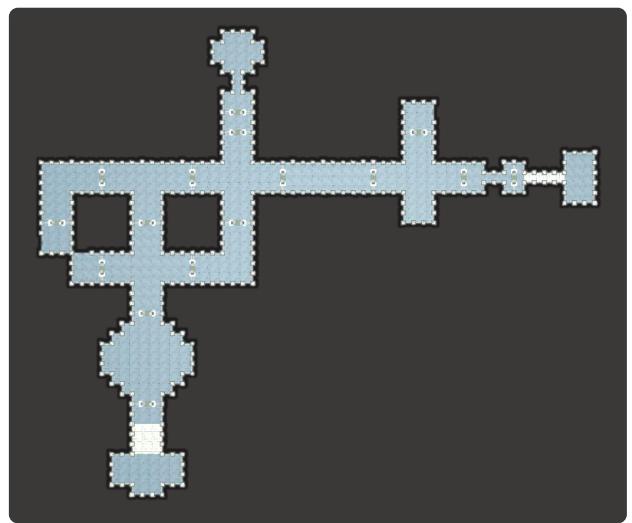
表 3.9: 空間替換表，出口端點與房間容器對應之範例

端點	對應之房間容器
Ex	

在圖 3.17 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) 使遊戲空間趨於完整，見圖 3.18。若增加空間替換表其對應的選項，便能創造出主線外的分支結構。



(a) Seed = 1



(b) Seed = 4

圖 3.18: 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 2.1.2 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 [5]，令地圖片段為房間容器，並進行改良以合適我方實驗環境。

圖 3.19 介紹地圖片段的演化流程。第一步驟，基於第 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第 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 3.19: 房間容器採取基因演算法之演化流程

3.3.1 基因的儲存結構

圖 3.20 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 3.20 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 3.20 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)。

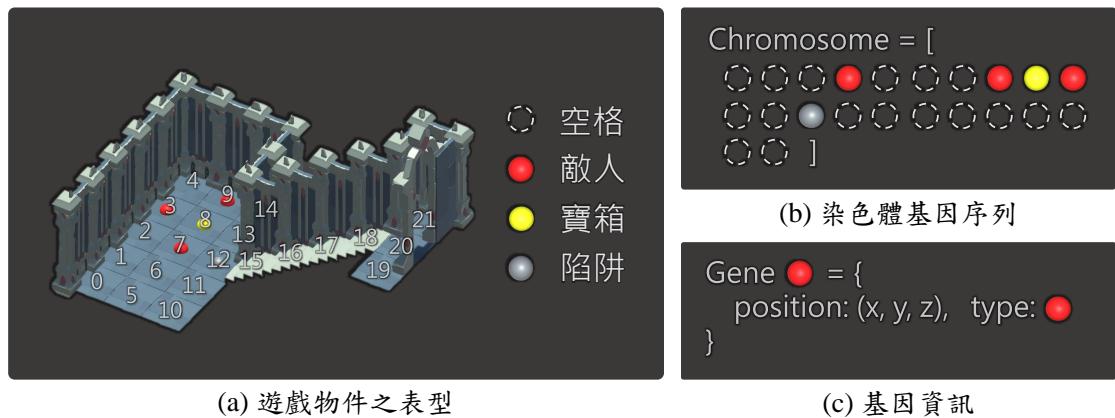


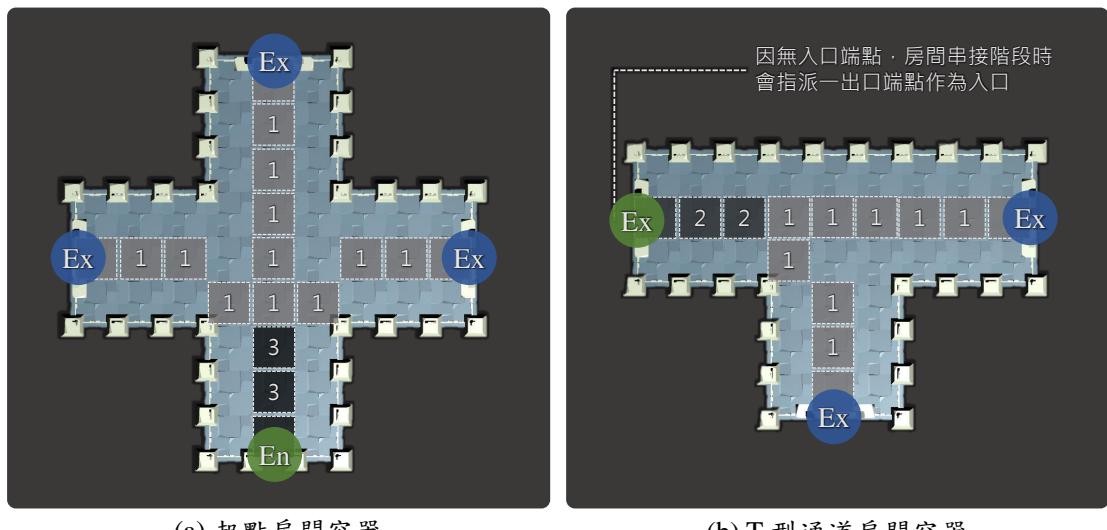
圖 3.20: 房間容器以遊戲物件的佈局方式作為染色體



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 3.10 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 3.10 a-b）。圖 3.21 使用 A-Star 搜尋演算法 [10] 建立行走空間，並搜素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。



(a) 起點房間容器

(b) T 型通道房間容器

圖 3.21: 動線於房間容器的計算方式，圖中數值為行經次數

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。

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



圖 3.22: 守衛點指標的遊戲體現情形

阻擋點 (Block)

方程式 3.2 為阻擋點的適應性函數。敵人會專注於阻擋玩家繼續前進，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 3.23 為阻擋點指標的實際遊玩情形，草綠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擋玩家前行。

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



圖 3.23: 阻擋點指標的遊戲體現情形

巡邏點 (Patrol)

方程式 3.3 為巡邏點的適應性函數。確保敵人擁有足夠的空間能夠進行移動，使其進行巡邏行為。 f_{ptl} 為 M 個敵人 (E_i) 其方圓半徑 (r) 內相鄰可行走瓦磚之數量總合，計算各個可行走瓦磚 (P_j) 與敵人的直線距離，若距離小於或等於 r 則值為 1，反之為 0。於本次實驗中，將採用 $r = 3$ 作為實驗範例，該數值可由遊戲設計師決定。圖 3.24 為巡邏點指標的實際遊玩情形，水藍色區塊為系統計算出的相鄰可走瓦磚，敵人會在此區域內執行巡邏的。

$$f_{ptl} = \sum_{i=1}^M neighbor(E_i, r) \quad (3.3)$$

$$neighbor(Obj, r) = \sum_{j=1}^N isNeighbor(Obj, P_j, r) \begin{cases} 1, & \text{if } dist(Obj, P_j) \leq r \\ 0, & \text{if } dist(Obj, P_j) > r \end{cases}$$



圖 3.24: 巡邏點指標的遊戲體現情形



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，函數皆有著一項共通點，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 3.4 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其餘權重絕對值之合，最小為 1， $w_{dst} = \max(1, \sum_{i=1}^{amount} |w_i|)$ 。

$$f_{dst} = \begin{cases} 1, & \text{if } N \leq O \leq M \\ 0, & \text{if } O < N \text{ or } O > M \end{cases} \quad (3.4)$$

3.3.4 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、阻擋點的權重調整至 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、阻擋點指標，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。

方程式 3.5 中定義房間容器（個體）的品質，參考 Abdullah Konak 提出的 *Weighted sum approaches* [11] 並予以改良。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 $\text{Normalized}(f_i) = [0, 1]$ ，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。

$$f_{all} = \sum_{i=1}^M (\text{Normalized}(f_i) \times w_i) \quad (3.5)$$

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

圖 3.25 中顯示了方程式 3.5 依照不同的自然數常數 c 進行適應值的標準化。在此圖 3.25 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。

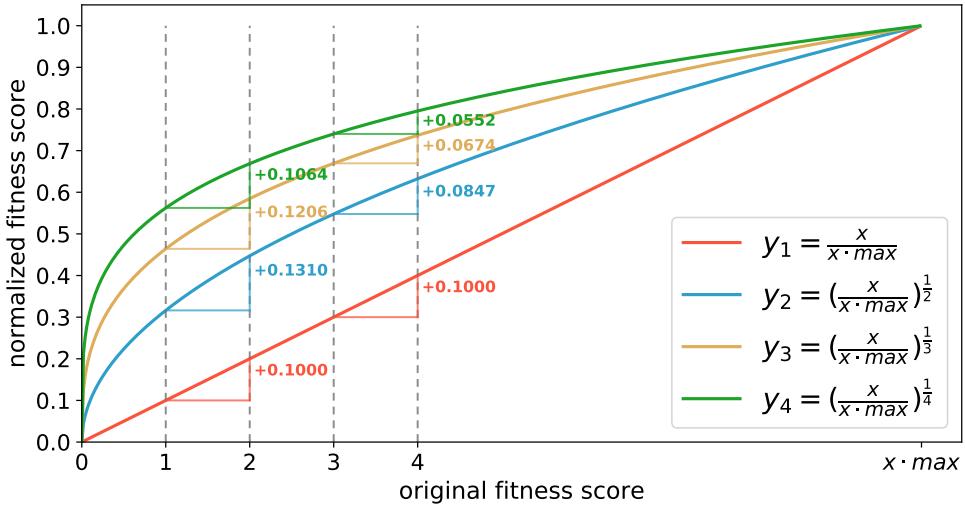


圖 3.25: 適應值進行標準化

舉例來說，承接圖 3.25 的原始標準化線性方程式如 y_1 ，並假設一個體 a 有守衛點適應值 $y_{guard_{1a}} = 0.2$ 與阻擋點適應值 $y_{block_{1a}} = 0.4$ ；另一個體 b 有守衛點適應值 $y_{guard_{1b}} = 0.0$ 與阻擋點適應值 $y_{block_{1b}} = 0.7$ 。在上述情形中，個體 a 的總適應值為 0.6、個體 b 的總適應值為 0.7，被列為最佳個體的會是個體 b 。但綜觀來看個體 a 的各個遊玩指標均衡發展，個體 a 應當是比個體 b 更能夠體現出複合型的遊玩指標。經過調整後的標準化線性方程式 y_2 ，個體 a 的總適應值為 $y_{guard_{2a}} + y_{block_{2a}} = 0.4472 + 0.6324 = 1.0796$ 、個體 b 的總適應值為 $y_{guard_{2b}} + y_{block_{2b}} = 0.0000 + 0.8366 = 0.8366$ ，此時能夠體現出複合型遊玩指標的個體 a 將會被視為是更優良的個體。

3.3.5 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 3.26。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。

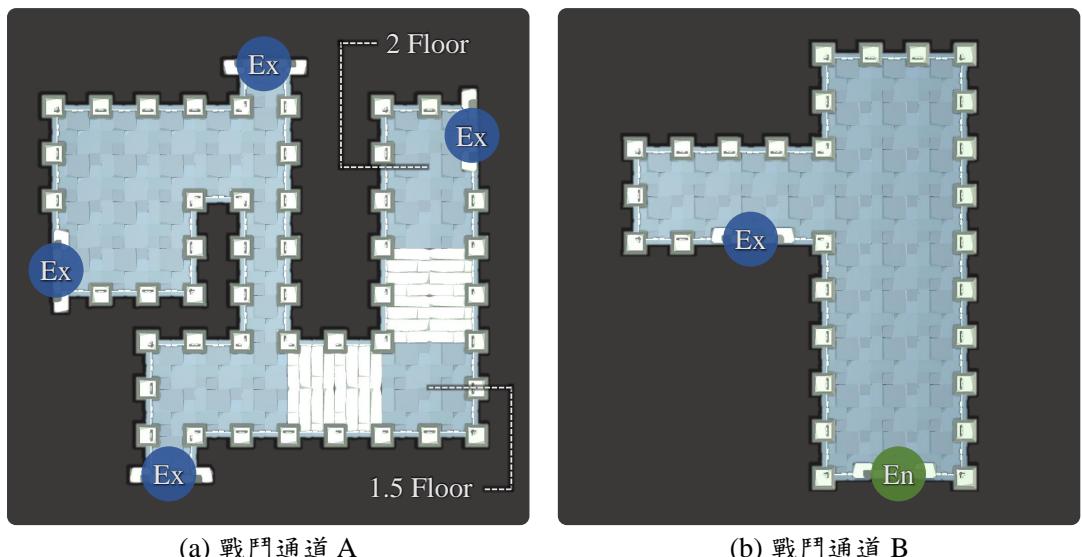


圖 3.26: 擴充房間容器，戰鬥通道的房間容器結構

寶藏房

寶藏房的端點識別物僅只有一個入口端點，便不存在空間動線，與空間動線相關的適應性函數不予採計。圖 3.27 是寶藏房的演化結果，圖 3.27 i-iii 的演化結果呈現出，敵人與寶箱物件彼此間的距離關係鄰近，敵人物件便能夠對相鄰寶箱物件進行守衛動作。

- 守衛點權重: 1.00
- 遊戲物件數量限制: [3, 5]

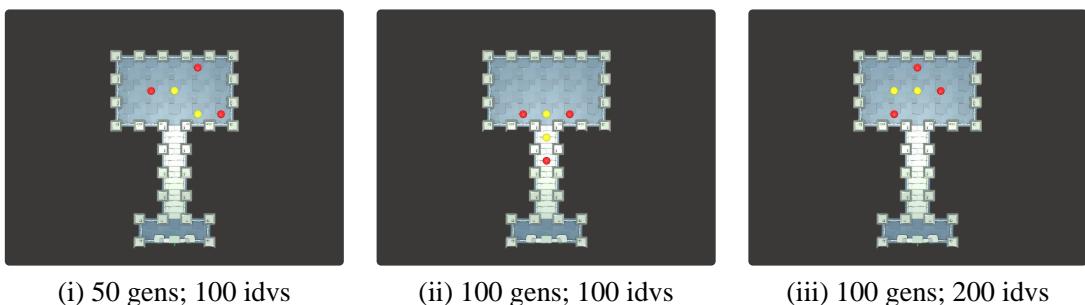


圖 3.27: 寶藏房的遊戲物件佈局演化示意

戰鬥通道（狹路驅逐）

戰鬥通道（狹路驅逐）無入口端點、有四個出口端點。在此房間容器中，西方出口為一廣場；東方出口位於二樓；在北方出口與南方出口之間有一狹長的通道，是連接各個出口的必經道路。

在此房間容器的佈局預期讓玩家體驗到重要幹道被敵人所盤據，玩家必須在該幹道上與敵人發生衝突方可通過，便將阻擋點指標之權重設為 1.00，並以南方出口作為入口。且在寬廣地區亦有機會設置敵人進行巡邏監視，便將巡邏點指標之權重設為 0.75。圖 3.28 是戰鬥通道（狹路驅逐）的演化結果，並以南方出口作為入口，圖 3.28 i-iii 能觀察到，並非所有的點都坐落在高權重的空間動線座標上，因為巡邏點指標的關係，將敵人物件放置於他處也能獲得相當不錯的部分適應值。

- 阻擋點權重: 1.00
- 巡邏點權重: 0.75
- 遊戲物件數量限制: [4, 5]



圖 3.28: 狹路驅逐情境 II 的遊戲物件佈局演化示意

戰鬥通道（鎮守要道）

戰鬥通道（鎮守要道）有一個入口端點及一個出口端點，在此房間容器的佈局與狹路驅逐近似，但敵人將專注於戰鬥陣行，將不會存在寶箱。圖 3.29 是戰鬥通道（鎮守要道）的演化結果，圖 3.29 i-iii 能夠觀察到演化結果中都不再存在寶箱物件。

- 阻擋點權重: 1.00
- 巡邏點權重: 0.50
- 守衛點權重: -1.00
- 遊戲物件數量限制: [3, 5]

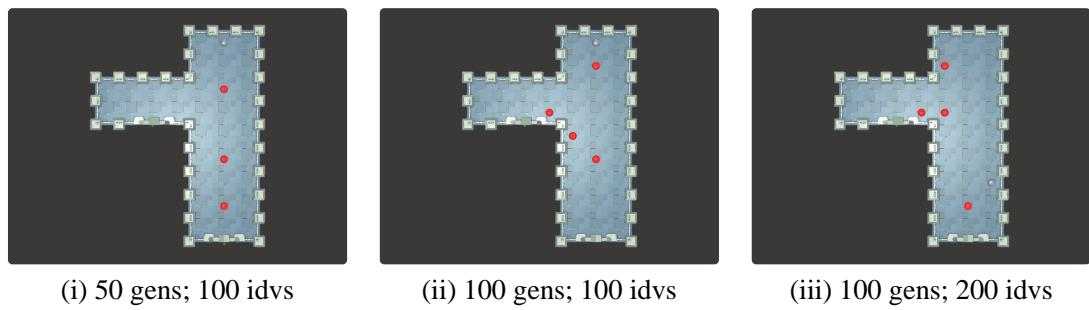


圖 3.29: 鎮守要道情境的遊戲物件佈局演化示意

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。圖 3.30 略過不需要進行演化的房間容器，並針對有設置遊玩特徵指標的房間容器進行演化。

3.4 方法彙整與總結

圖 3.31 將上述所提及之方法彙整至其中，共計三大項目。第 3.1 節介紹了「任務語法」，讓關卡設計師能透過抽象宏觀的概念，藉由直覺的圖形介面生成出任務圖，表達出遊玩流程的拓撲結構。第 3.2 節介紹了「空間建構」，從建構房間容器的最小體素單位至端點識別物，並且透過改寫系統將任務圖轉換至遊戲空間。第 3.3 節介紹了「地圖片段演化方法」，將房間容器依照不同的戰略要素，給予不同遊玩特徵指標的權重，最後利用基因演算法逐步演化得到遊戲物件的關卡最適佈局。

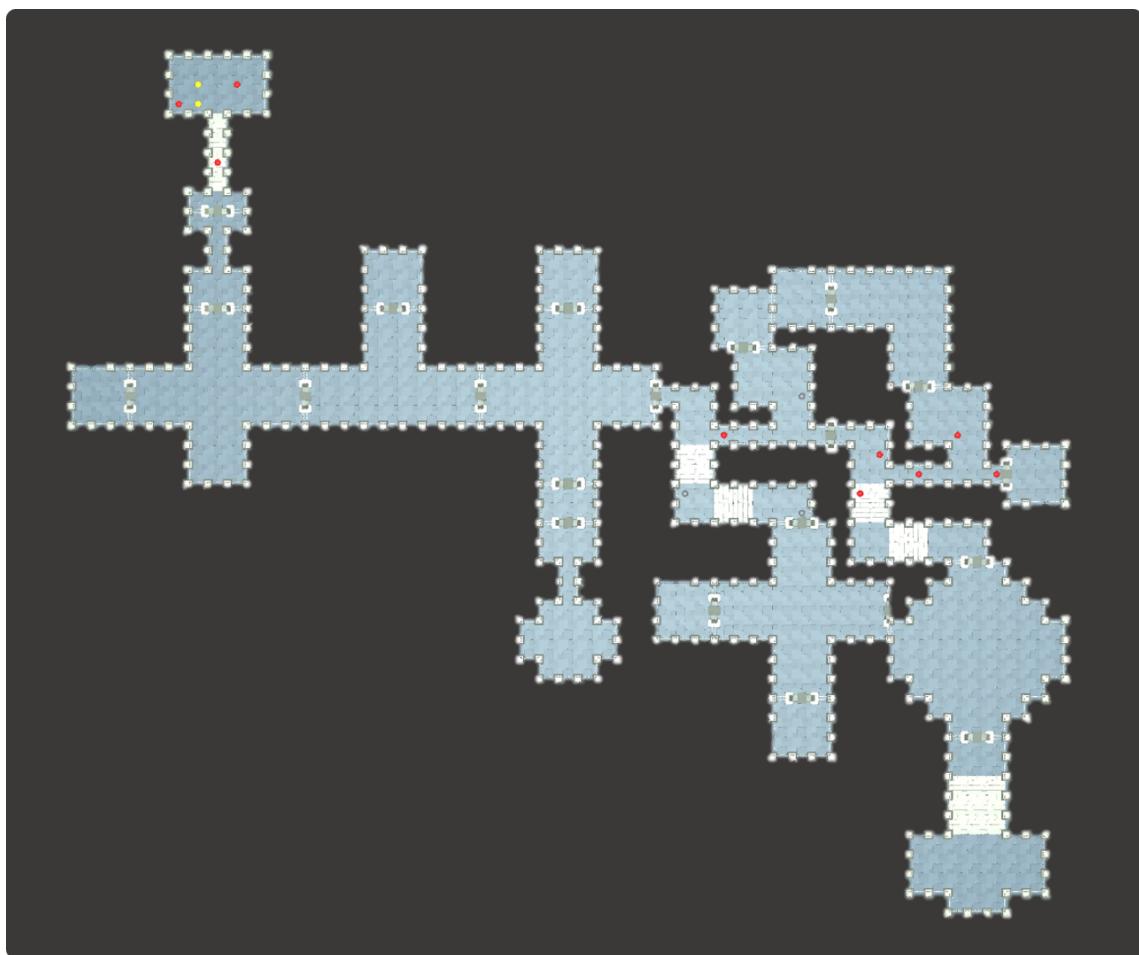


圖 3.30: 將關卡中全部房間容器進行遊戲物件的佈局演化

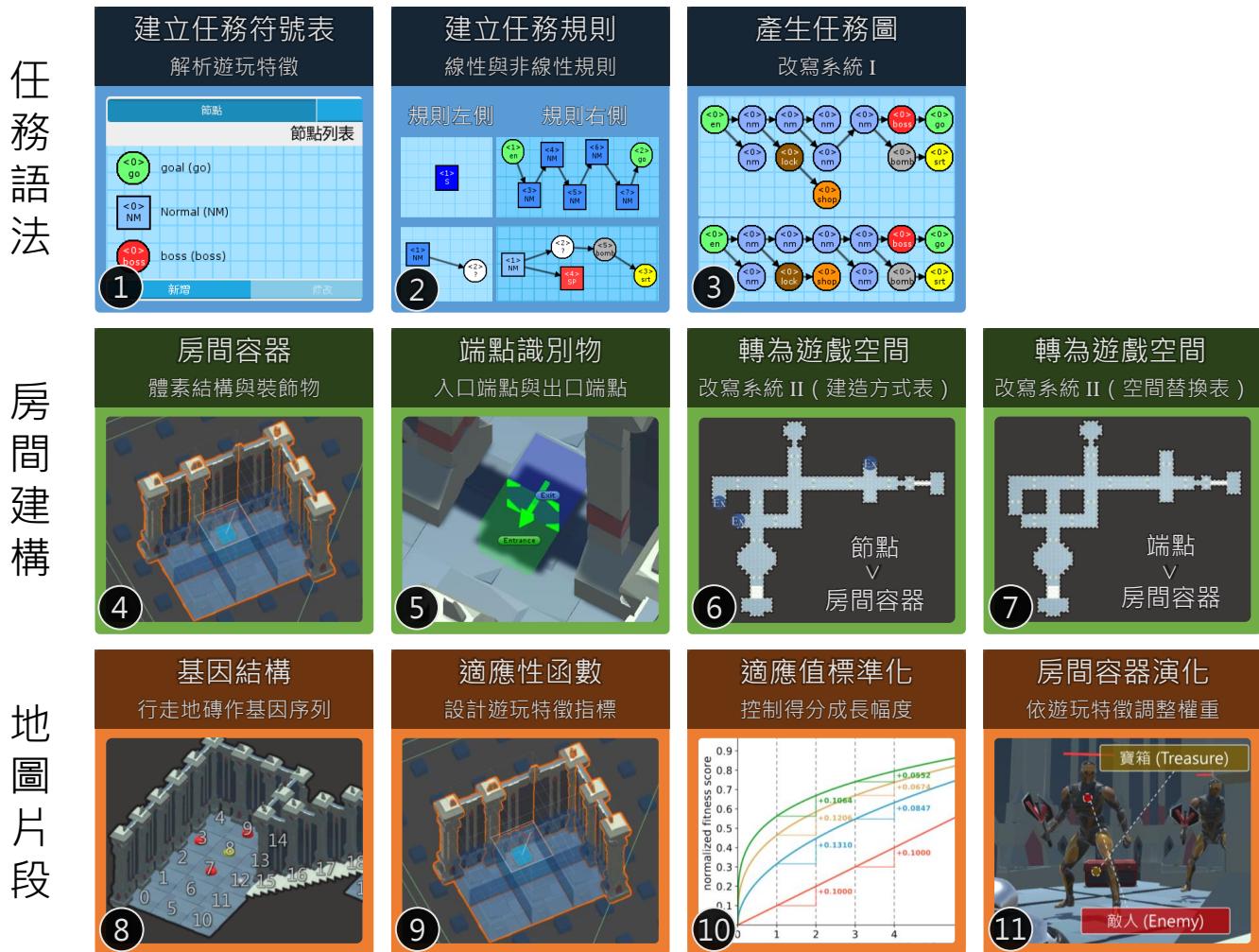


圖 3.31: 本論文提出之方法，完整框架示意圖

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：第 4.1 節，對本次實驗之環境建置、演算法使用之參數進行解釋；第 4.2 節，描述本次研究實驗之各項步驟；第 4.3 節，針對所挑選的房間採用不同權重的適應性函數，其族群大小會如何影響房間內的遊戲物件之佈局結果。

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

敵人會在玩家靠近，且進入攻擊視野範圍時，追擊玩家。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。當玩家碰觸到陷阱

時，會受到一定程度的損傷。

4.1.3 實驗參數設定與名詞解釋

本次實驗中將專注於檢視遊戲物件佈局的結果，而任務語法將沿用圖 3.16 所生成之任務圖，為使實驗結果清晰易讀，將移除無關聯之房間容器於實驗範例。房間容器的遊戲物件佈局演化階段時，每一次世代的演化過程有 80% 機率於父母代進行兩點交配 (two-point crossover)；10% 機率令衍生子代會進行突變，個體的染色體有 5% 至 20% 的基因會轉換成其它的遊戲物件種類，設定值參考自 [5]。相關實驗所使用之名詞，請參照表 4.1 之釋義。

然而各個房間容器的適應性函數之權重如何決定？將取決於關卡設計師對於該房間容器預期產出進行調整，若希望該遊玩特徵指標能愈出現於房間容器，則將其權重調高 $w = (0, 1]$ ；反之，愈不希望該遊玩特徵指標出現在房間容器，則將其權重調低 $w = [-1, 0)$ ；若不理會該遊玩特徵指標的體現與否，則將其權重設定為 $w = 0$ 。在本次研究的系列實驗中，將以關卡設計師視角進行權重的指派，並不代表實驗中所使用的權重最能夠體現其房間容器。

表 4.1: 實驗使用之名詞釋義

名詞	名詞解釋
回合 (run)	回合為使用相同參數進行的實驗，但實驗間的演化過程與結果是彼此獨立進行。
世代 (generation)	世代為完整地完成一輪的演化，包含了擇優交配、子代突變與世代交替。
族群 (population)	族群為每世代中個體的數量。
個體 (individual)	個體為染色體集合的單位，在本次實驗環境中個體僅有一條染色體，該染色體為遊戲物件佈局的基因序列。
相鄰	周圍是特定一遊戲物件，其相距僅一格正四方或一格斜四方的距離範圍內的其它遊戲物件。
標準化加權適應值	原始的適應值進行標準化處理後，並依照對應權重值進行加權，最後得到一數值 $score = [-1, 1]$ 。

4.2 實驗流程

4.2.1 透過任務語法建立任務圖

第一步驟，藉由第 3.1 節所介紹的任務語法與改寫系統 I，搭配任務符號表（表 3.1、3.2）以及任務規則表（表 3.3、3.4、3.5、3.6），進行任務圖的生成工作。

4.2.2 基於任務圖轉換為遊戲空間

第二步驟，基於前一小節所生成的任務圖（圖 XXXXXX），搭配第 3.2 節所建置的房間容器（圖 3.12、3.13、3.14、3.15），與其對應的建造方式表（表 3.8）與空間替換表（表 3.9），將該任務圖轉換為遊戲空間。



4.2.3 資料收集

第三步驟，將收集第 3.3 節中，房間容器進行基因演算法演化時，於演化過程中收集下述資訊並輸出 CSV 格式資料：

- 紀錄各回合、世代中，其各個個體（單一染色體）的適應值經標準化後的得分狀況，見表 4.2。
- 紀錄各回合、世代中，其各個個體（單一染色體）的所有座標資訊與其遊戲物件類型，見表 4.3。
- 在每一回合結束時，會輸出其演化運算之耗時。更多的原始資料節錄內容可參見附錄資源。

4.2.4 房間容器演化

4.3 房間容器佈局演化之結果

在第 3.3.5 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

表 4.2: 演化適應值資料節錄示意

回合編號	世代編號	個體編號	指標	標準化加權適應值
1	1	1	阻擋點	0
1	1	1	巡邏點	0
1	1	1	守衛點	0
1	1	1	遊戲物件數量	0
1	1	2	阻擋點	0.6540
1	1	2	巡邏點	0.5587

表 4.3: 演化座標資料節錄示意

回合編號	世代編號	個體編號	座標	遊戲物件類型
1	1	1	(1.0, 1.0, 0.0)	Treasure
1	1	1	(0.0, 1.0, 0.0)	Empty
1	1	1	(0.0, 1.0, 2.0)	Empty
1	1	2	(1.0, 1.0, 0.0)	Enemy
1	1	2	(0.0, 1.0, 0.0)	Empty

多樣性的遊戲物件佈局。在本小節的各實驗中，將針對上述三種房間容器的演化結果做更進一步的分析，紀錄佈局演化結果上視圖與最佳個體之標準化加權適應值，由於各實驗、各回合產生的區域極值不同，影響標準化的量尺不同，我們便分開探討各實驗中的適應值成長趨勢。

4.3.1 寶藏房

於本小節中，展示了寶藏房使用三種不同參數的演化結果，表 4.4、4.6 與 4.8 為採用的參數設定，其中演化世代數量皆為 100 次，族群大小分別為 50、100 與 200 個個體，表 4.5、4.7 與 4.9 分別為其演化結果的最佳個體適應值的得分情形。

在實驗開始進行前能夠預想到，期望寶藏房能夠體現出守衛點的特性，最理想的守衛點佈局便是達到遊戲物件數量上限 5 個的當下，所有的遊戲物件皆由敵人與寶箱所組成，不存在其它的遊戲物件。我們觀察到圖 4.1（第一實驗組）的 ix 演化結果中含有一個陷阱物件，而在圖 4.2（第二實驗組）及圖 4.3（第三實驗組）皆沒有任何陷阱物件被生成。

圖 4.1（第一實驗組）的演化結果中，十個不同回合皆存在著寶箱物件相鄰敵人物件的遊玩特徵。但圖 4.1 vi 與 vii 中，發現當中各有一個寶箱的周圍沒有任何敵人物件（圖 vi，廣場東南方向；圖 vii，階梯上），這對於守衛點的設計理念來說，這並不是一個優良的特徵。

綜合上述兩段落的現象，我們提出一個假說，族群大小有可能會導致演化過程陷入局部最佳解的困境。在演化耗時上，族群大小幾乎是與演化耗時成等比例成長的關係，因此考量到玩家在進行遊戲時的遊玩體驗，遊戲進行演化的讀取時間與演化族群的大小之間必須取得一平衡。



表 4.4: 實驗 Treasure-1 之基因演算法參數配置

回合次數	世代數量	個體數量
10	100	50

指標	權重	備注
守衛點	1.00	
遊戲物件數量	1.00	<i>Limit : [2, 5]</i>

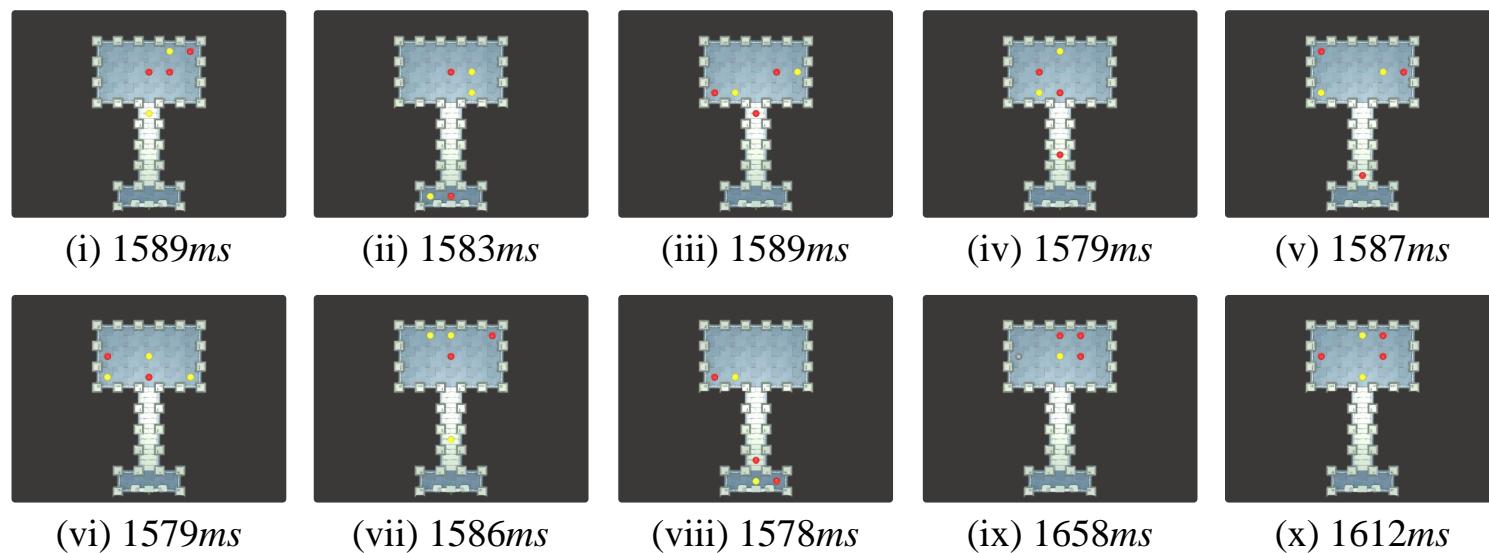


圖 4.1: 實驗 Treasure-1 的演化結果及各演化耗時

表 4.5: 實驗 Treasure-1 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	守衛點	0.6008	1
	物件數量	1.0000	1
	1.6008		
2	守衛點	0.6670	1
	物件數量	1.0000	1
	1.6670		
3	守衛點	0.8340	1
	物件數量	1.0000	1
	1.8340		
4	守衛點	0.8021	1
	物件數量	1.0000	1
	1.8021		
5	守衛點	0.5524	1
	物件數量	1.0000	1
	1.5524		
回合	指標	適應值	權重
	總適應值		
6	守衛點	0.8045	1
	物件數量	1.0000	1
	1.8045		
7	守衛點	0.6299	1
	物件數量	1.0000	1
	1.6299		
8	守衛點	0.5853	1
	物件數量	1.0000	1
	1.5853		
9	守衛點	0.6315	1
	物件數量	1.0000	1
	1.6315		
10	守衛點	0.5439	1
	物件數量	1.0000	1
	1.5439		

表 4.6: 實驗 Treasure-2 之基因演算法參數配置

回合次數	世代數量	個體數量
10	100	100

指標	權重	備注
守衛點	1.00	
遊戲物件數量	1.00	<i>Limit : [2, 5]</i>

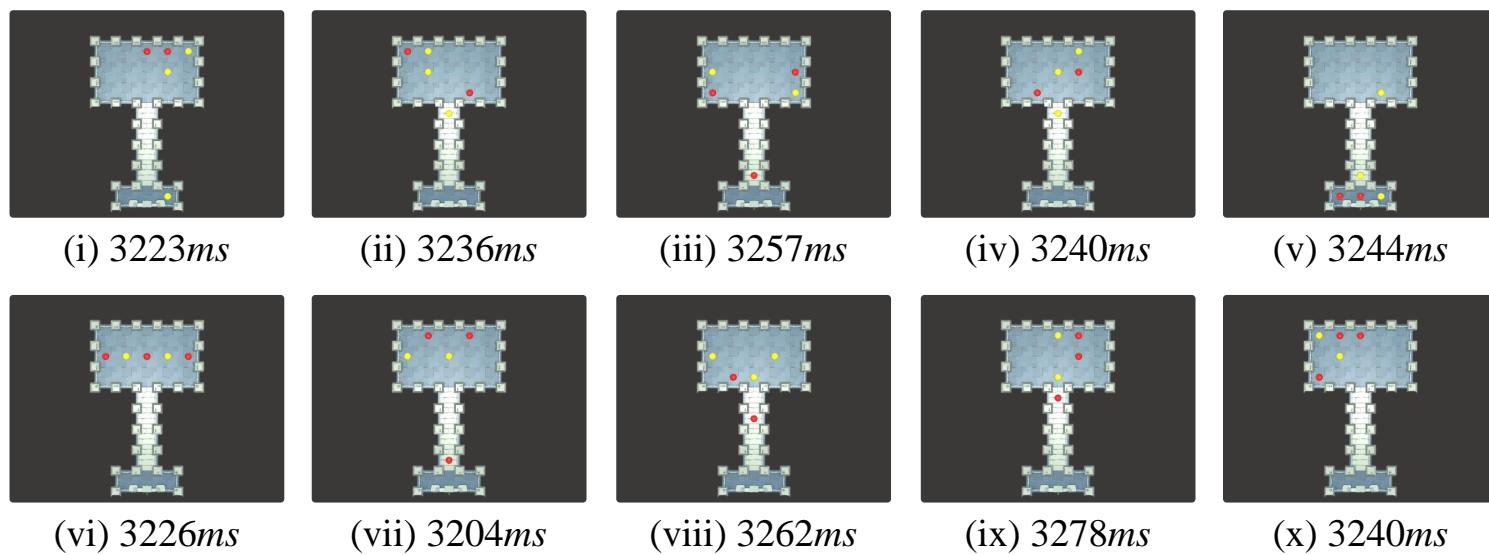


圖 4.2: 實驗 Treasure-2 的演化結果及各演化耗時

表 4.7: 實驗 Treasure-2 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	守衛點	0.7806	1
	物件數量	1.0000	1
	1.7806		
2	守衛點	0.5759	1
	物件數量	1.0000	1
	1.5759		
3	守衛點	0.6757	1
	物件數量	1.0000	1
	1.6757		
4	守衛點	0.7658	1
	物件數量	1.0000	1
	1.7658		
5	守衛點	0.6024	1
	物件數量	1.0000	1
	1.6024		
回合	指標	適應值	權重
	總適應值		
6	守衛點	0.6795	1
	物件數量	1.0000	1
	1.6795		
7	守衛點	0.6330	1
	物件數量	1.0000	1
	1.6331		
8	守衛點	0.6504	1
	物件數量	1.0000	1
	1.6504		
9	守衛點	0.6753	1
	物件數量	1.0000	1
	1.6753		
10	守衛點	0.7028	1
	物件數量	1.0000	1
	1.7028		

表 4.8: 實驗 Treasure-3 之基因演算法參數配置

回合次數	世代數量	個體數量
10	100	200

指標	權重	備注
守衛點	1.00	
遊戲物件數量	1.00	<i>Limit : [2, 5]</i>

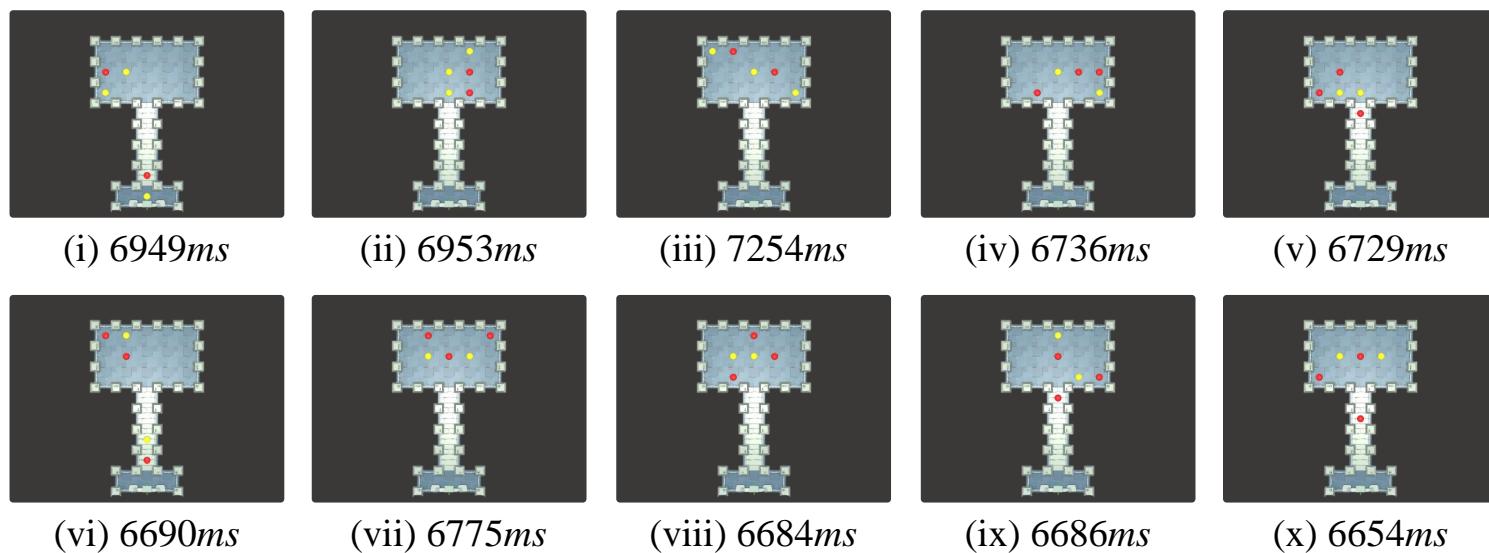


圖 4.3: 實驗 Treasure-3 的演化結果及各演化耗時

表 4.9: 實驗 Treasure-3 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	守衛點	0.5652	1
	物件數量	1.0000	1
	1.5652		
2	守衛點	0.7938	1
	物件數量	1.0000	1
	1.7938		
3	守衛點	0.8965	1
	物件數量	1.0000	1
	1.8965		
4	守衛點	0.6921	1
	物件數量	1.0000	1
	1.6921		
5	守衛點	0.6988	1
	物件數量	1.0000	1
	1.6988		
回合	指標	適應值	權重
	總適應值		
6	守衛點	0.7351	1
	物件數量	1.0000	1
	1.7351		
7	守衛點	0.7051	1
	物件數量	1.0000	1
	1.7051		
8	守衛點	0.6047	1
	物件數量	1.0000	1
	1.6047		
9	守衛點	0.6483	1
	物件數量	1.0000	1
	1.6483		
10	守衛點	0.7545	1
	物件數量	1.0000	1
	1.7545		

4.3.2 戰鬥通道（狹路驅逐）

於本小節中，展示了戰鬥通道（狹路驅逐）使用三種不同參數的演化結果，表 4.10、4.12 與 4.14 為採用的參數設定，其中演化世代數量皆為 100 次，族群大小分別為 50、100 與 200 個個體，表 4.11、4.13 與 4.15 分別為其演化結果的最佳個體適應值的得分情形。

為了驗證於前一小節提出的假說，將觀察規模較小的族群大小情形下，是否吻合局部最佳解的特徵。在圖 4.4（第一實驗組）的 iv 中可見，遊戲物件的數量遠遠超乎我們所規定的範疇內，更進一步的查看其配分情形，於表 4.11 中可得知該次演化結果的阻擋點為 1.0000 分與巡邏點 0.7500 分，二種遊玩指標皆取得了滿分的適應值，推估於該次演化世代中，所有的個體演化後皆已超過限制的個體數量範圍，導致無法順利獲得個體數量的平衡適應值分數 1.7500 分。因含有機率的性質，雷同的情形亦會發生於較大族群大小的案例中，但發生的機會明顯會少上許多。



表 4.10: 實驗 Narrow-1 之基因演算法參數配置

回合次數	世代數量	個體數量
10	100	50

指標	權重	備注
阻擋點	1.00	
巡邏點	0.75	
遊戲物件數量	1.75	<i>Limit : [4, 5]</i>

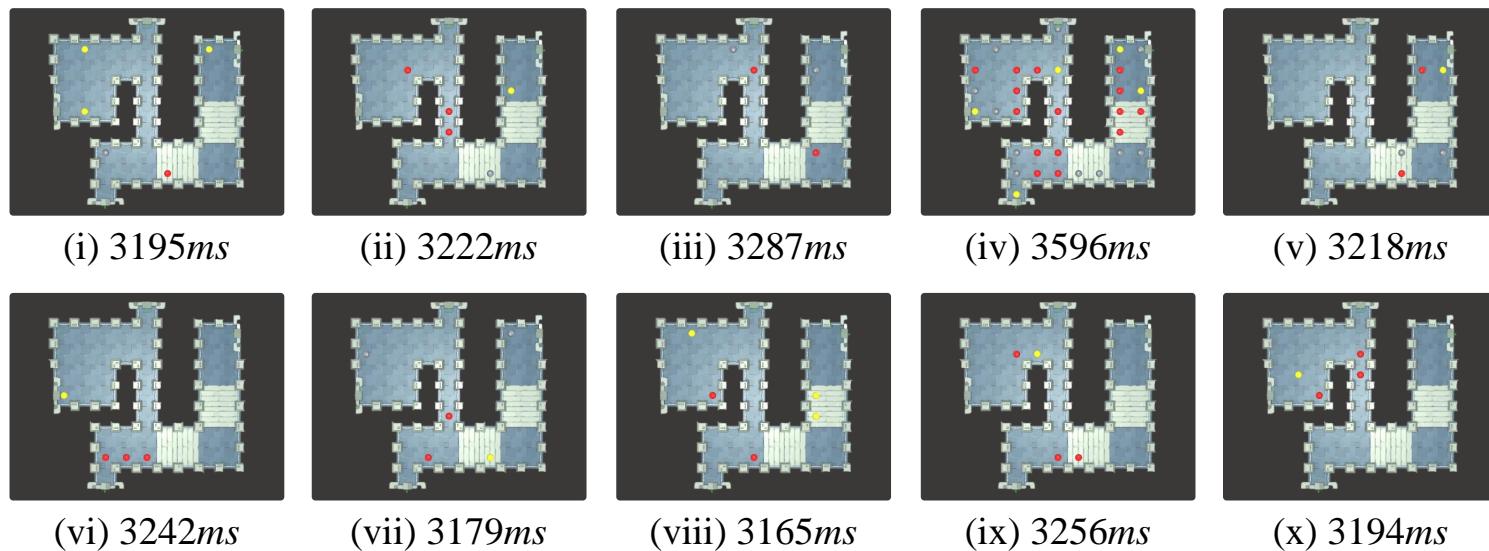


圖 4.4: 實驗 Narrow-1 的演化結果及各演化耗時

表 4.11: 實驗 Narrow-1 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	阻擋點	0.4082	1
	巡邏點	0.6708	0.75
	物件數量	1.7500	1.75
	2.8291		
2	阻擋點	0.7906	1
	巡邏點	0.7310	0.75
	物件數量	1.7500	1.75
	3.2716		
3	阻擋點	0.5345	1
	巡邏點	0.6944	0.75
	物件數量	1.7500	1.75
	2.9789		
4	阻擋點	1.0000	1
	巡邏點	0.7500	0.75
	物件數量	0.0000	1.75
	1.7500		
5	阻擋點	0.4472	1
	巡邏點	0.5669	0.75
	物件數量	1.7500	1.75
	2.7642		
6	阻擋點	0.8660	1
	巡邏點	0.6124	0.75
	物件數量	1.7500	1.75
	3.2284		
7	阻擋點	0.9129	1
	巡邏點	0.5929	0.75
	物件數量	1.7500	1.75
	3.2558		
8	阻擋點	0.7559	1
	巡邏點	0.6204	0.75
	物件數量	1.7500	1.75
	3.1263		
9	阻擋點	1.0000	1
	巡邏點	0.7206	0.75
	物件數量	1.7500	1.75
	3.4706		
10	阻擋點	0.9129	1
	巡邏點	0.7500	0.75
	物件數量	1.7500	1.75
	3.4129		

表 4.12: 實驗 Narrow-2 之基因演算法參數配置

回合次數	世代數量	個體數量
10	100	100

指標	權重	備注
阻擋點	1.00	
巡邏點	0.75	
遊戲物件數量	1.75	<i>Limit : [4, 5]</i>

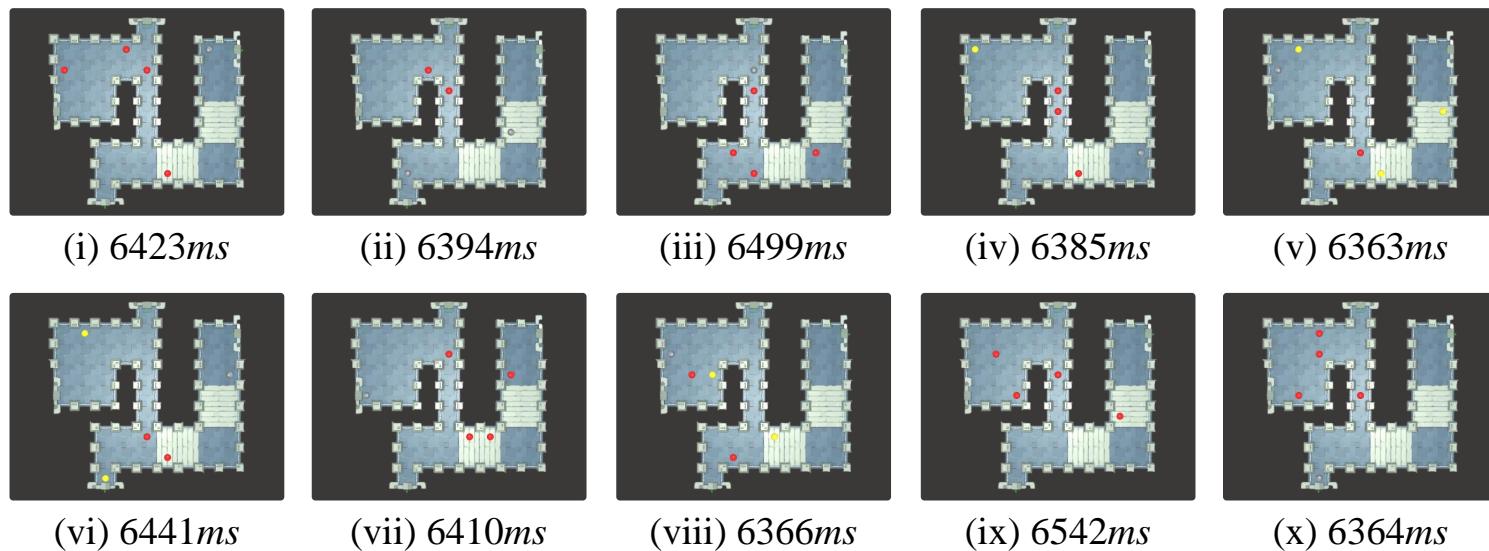


圖 4.5: 實驗 Narrow-2 的演化結果及各演化耗時

表 4.13: 實驗 Narrow-2 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	阻擋點	0.7746	1
	巡邏點	0.7227	0.75
	物件數量	1.7500	1.75
	3.2473		
2	阻擋點	0.7746	1
	巡邏點	0.7500	0.75
	物件數量	1.7500	1.75
	3.2746		
3	阻擋點	0.7906	1
	巡邏點	0.7500	0.75
	物件數量	1.7500	1.75
	3.2906		
4	阻擋點	0.7454	1
	巡邏點	0.6495	0.75
	物件數量	1.7500	1.75
	3.1449		
5	阻擋點	0.5774	1
	巡邏點	0.7262	0.75
	物件數量	1.7500	1.75
	3.0535		
6	阻擋點	0.7746	1
	巡邏點	0.7500	0.75
	物件數量	1.7500	1.75
	3.2746		
7	阻擋點	0.7559	1
	巡邏點	0.5929	0.75
	物件數量	1.7500	1.75
	3.0989		
8	阻擋點	0.7071	1
	巡邏點	0.6124	0.75
	物件數量	1.7500	1.75
	3.0695		
9	阻擋點	0.8944	1
	巡邏點	0.7500	0.75
	物件數量	1.7500	1.75
	3.3944		
10	阻擋點	0.8165	1
	巡邏點	0.7016	0.75
	物件數量	1.7500	1.75
	3.2681		

表 4.14: 實驗 Narrow-3 之基因演算法參數配置

回合次數	世代數量	個體數量
10	100	200

指標	權重	備注
阻擋點	1.00	
巡邏點	0.75	
遊戲物件數量	1.75	<i>Limit : [4, 5]</i>

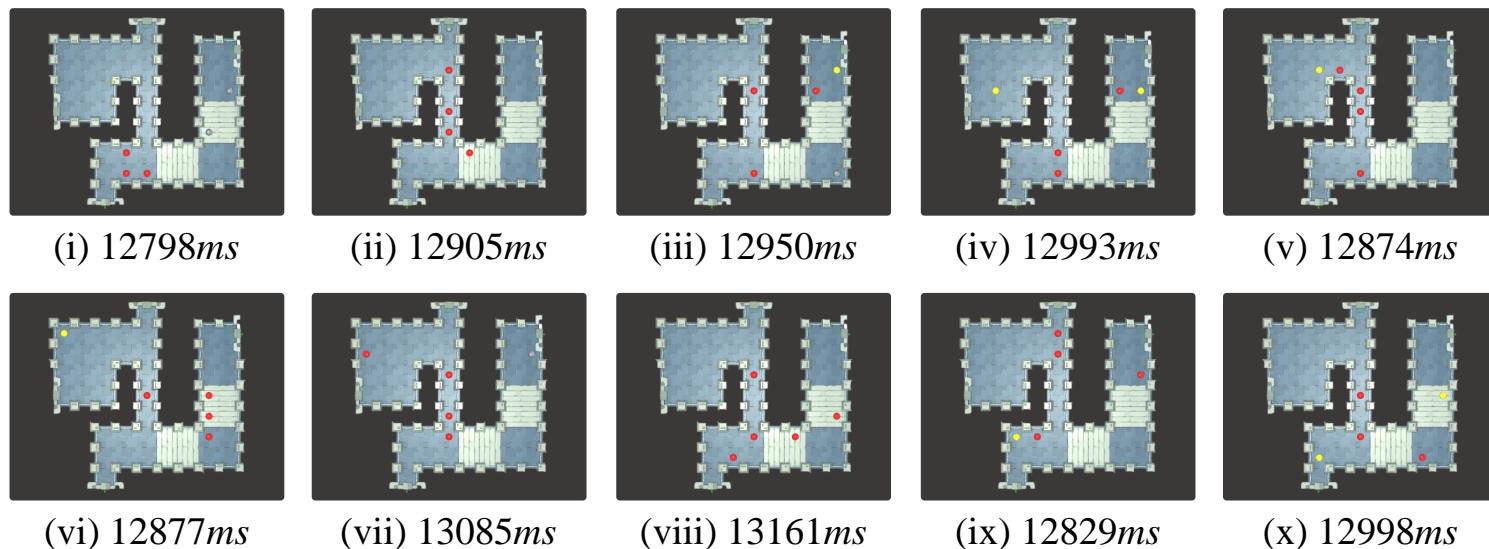


圖 4.6: 實驗 Narrow-3 的演化結果及各演化耗時

表 4.15: 實驗 Narrow-3 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	阻擋點	0.8660	1
	巡邏點	0.6982	0.75
	物件數量	1.7500	1.75
	3.3142		
2	阻擋點	0.8165	1
	巡邏點	0.7016	0.75
	物件數量	1.7500	1.75
	3.2681		
3	阻擋點	0.8660	1
	巡邏點	0.6339	0.75
	物件數量	1.7500	1.75
	3.2499		
4	阻擋點	0.8660	1
	巡邏點	0.6124	0.75
	物件數量	1.7500	1.75
	3.2284		
5	阻擋點	1.0000	1
	巡邏點	0.6047	0.75
	物件數量	1.7500	1.75
	3.3547		
回合	指標	適應值	權重
	總適應值		
6	阻擋點	0.8165	1
	巡邏點	0.6495	0.75
	物件數量	1.7500	1.75
	3.2160		
7	阻擋點	0.7746	1
	巡邏點	0.6760	0.75
	物件數量	1.7500	1.75
	3.2006		
8	阻擋點	0.9428	1
	巡邏點	0.7500	0.75
	物件數量	1.7500	1.75
	3.4428		
9	阻擋點	0.7071	1
	巡邏點	0.6760	0.75
	物件數量	1.7500	1.75
	3.1331		
10	阻擋點	0.8944	1
	巡邏點	0.5441	0.75
	物件數量	1.7500	1.75
	3.1885		

4.3.3 戰鬥通道（鎮守要道）

於本小節中，展示了戰鬥通道（鎮守要道）使用三種不同參數的演化結果，表 4.17、4.19 與 4.21 為採用的參數設定，其中演化世代數量皆為 100 次，族群大小分別為 50、100 與 200 個個體，表 4.18、4.20 與 4.22 分別為其演化結果的最佳個體適應值的得分情形。

在這一小節中，我們刻意將守衛點的權重調整為 1。在圖 4.7（第一實驗組）的 i 與 viii 中，仍出現了寶箱的物件；且 vii 的敵人皆不在空間動線上進行阻擋行為。觀察到表 4.18 中，前述三個回合的演化適應值總分與其它回合相比遜色許多，藉此也驗證了平衡適應性函數的可靠性，依賴其高權重的緣故，能夠限制在絕大多數的情形下，確保演化結果落於限制範圍內。

隨著族群大小的提升，我們在圖 4.8（第二實驗組）與圖 4.9（第三實驗組），已看不見與未採用正權重適應性函數相關遊戲物件（寶箱），且各回合最佳個體的適應值也愈趨於平均，總體標準差 σ 也愈來愈小，見表 4.16。

表 4.16: 戰鬥通道（鎮守要道）演化結果之平均適應值與總體標準差 σ

族群大小	平均適應值	總體標準差 σ
50	3.6347	0.42099
100	3.7713	0.29068
200	3.8458	0.12758

表 4.17: 實驗 Trunk-1 之基因演算法參數配置

回合次數	世代數量	個體數量
1	100	50

指標	權重	備注
阻擋點	1.00	
巡邏點	0.50	
守衛點	-1.00	
遊戲物件數量	2.50	<i>Limit : [3, 5]</i>

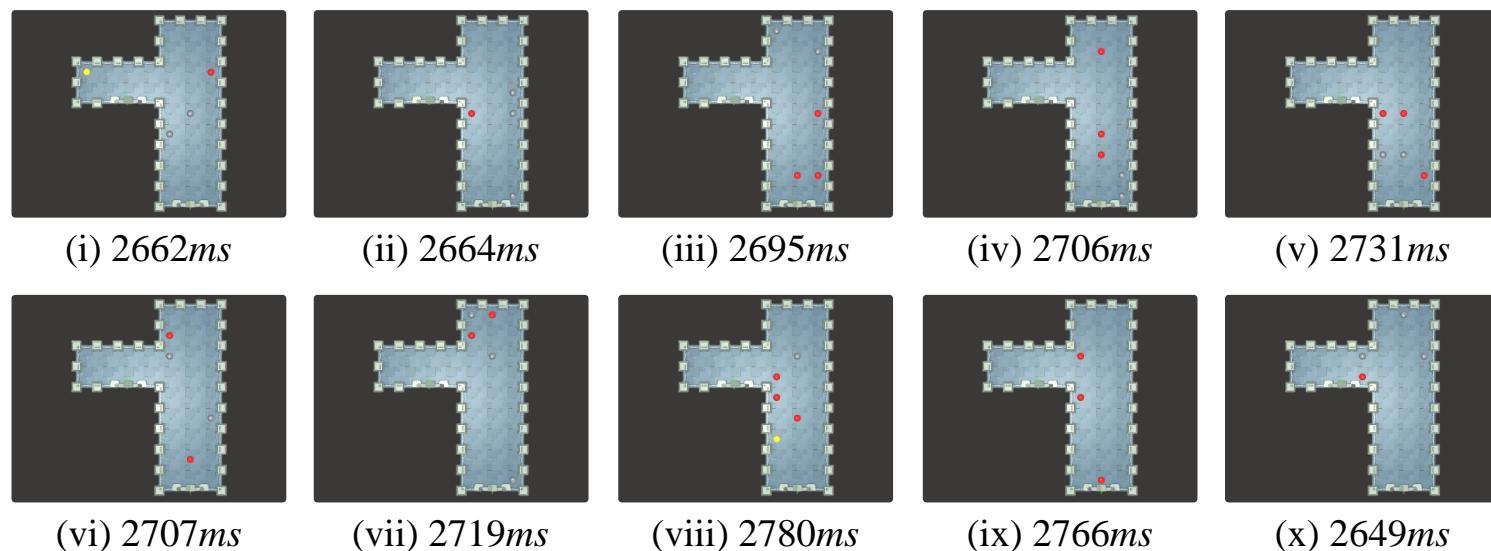


圖 4.7: 實驗 Trunk-1 的演化結果及各演化耗時

表 4.18: 實驗 Trunk-1 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	阻擋點	0.0000	1
	巡邏點	0.4730	0.5
	守衛點	-0.1925	-1
	物件數量	2.5000	2.5
2.7804			
2	阻擋點	1.0000	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
4.0000			
3	阻擋點	1.0000	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
4.0000			
4	阻擋點	0.7071	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
3.7071			
5	阻擋點	0.7071	1
	巡邏點	0.4541	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
3.6613			
回合	指標	適應值	權重
	總適應值		
6	阻擋點	1.0000	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
4.0000			
7	阻擋點	0.0000	1
	巡邏點	0.4320	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
2.9320			
8	阻擋點	1.0000	1
	巡邏點	0.4970	0.5
	守衛點	-0.4307	-1
	物件數量	2.5000	2.5
3.5663			
9	阻擋點	1.0000	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
4.0000			
10	阻擋點	0.7071	1
	巡邏點	0.4932	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
3.7003			

表 4.19: 實驗 Trunk-2 之基因演算法參數配置

回合次數	世代數量	個體數量
1	100	100

指標	權重	備注
阻擋點	1.00	
巡邏點	0.50	
守衛點	-1.00	
遊戲物件數量	2.50	<i>Limit : [3, 5]</i>

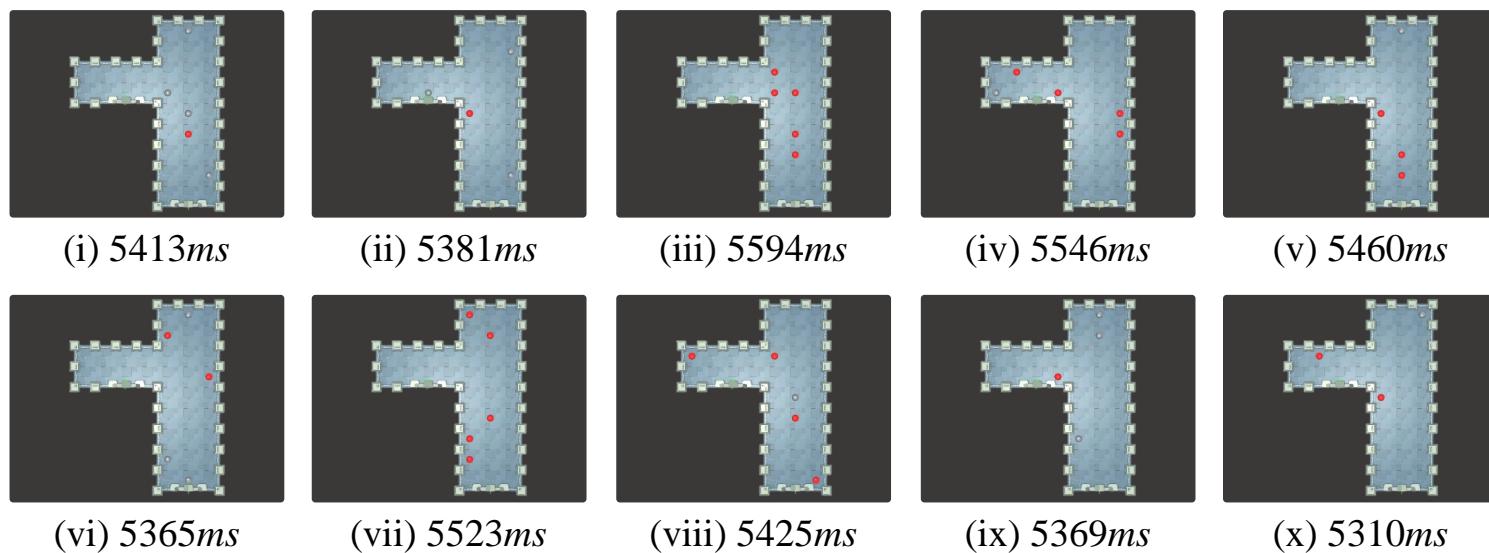


圖 4.8: 實驗 Trunk-2 的演化結果及各演化耗時

表 4.20: 實驗 Trunk-2 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	阻擋點	1.0000	1
	巡邏點	0.4743	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.9743		
2	阻擋點	0.7071	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.7071		
3	阻擋點	0.8165	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.8165		
4	阻擋點	0.7071	1
	巡邏點	0.4872	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.6943		
5	阻擋點	1.0000	1
	巡邏點	0.4552	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.9552		
回合	指標	適應值	權重
	總適應值		
6	阻擋點	0.0000	1
	巡邏點	0.4737	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	2.9737		
7	阻擋點	1.0000	1
	巡邏點	0.4806	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.9806		
8	阻擋點	0.7071	1
	巡邏點	0.4887	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.6958		
9	阻擋點	1.0000	1
	巡邏點	0.4932	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.9932		
10	阻擋點	1.0000	1
	巡邏點	0.4226	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.9226		

表 4.21: 實驗 Trunk-3 之基因演算法參數配置

回合次數	世代數量	個體數量
1	100	200

指標	權重	備注
阻擋點	1.00	
巡邏點	0.50	
守衛點	-1.00	
遊戲物件數量	2.50	<i>Limit : [3, 5]</i>

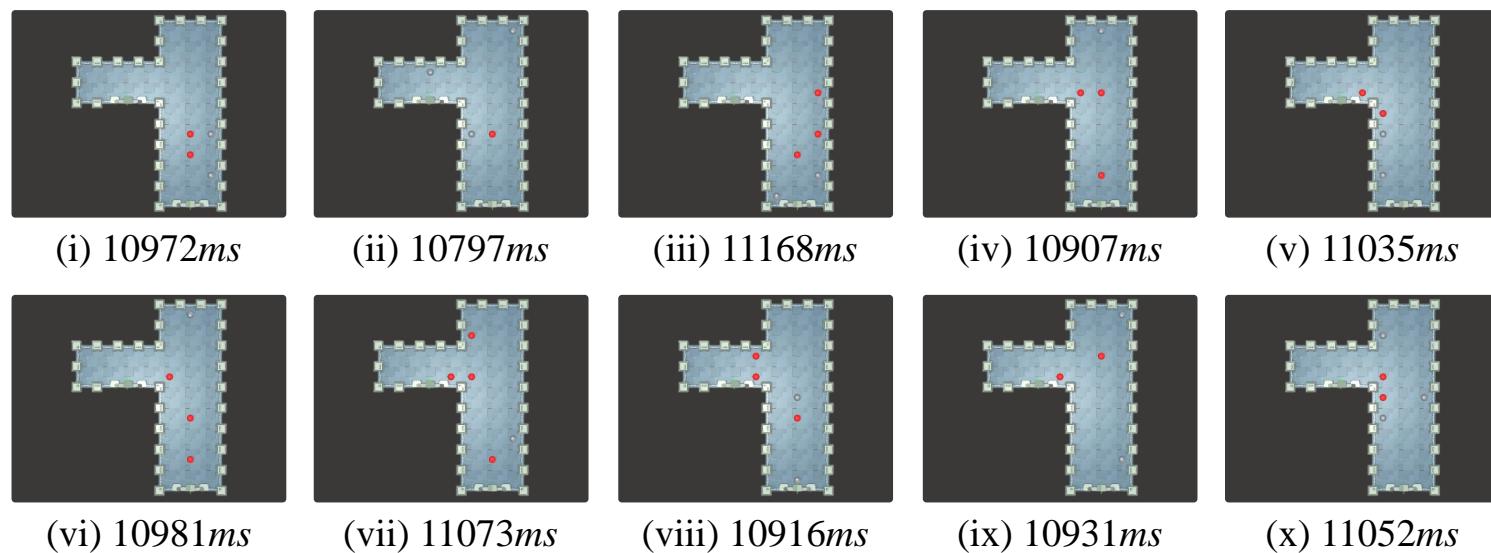


圖 4.9: 實驗 Trunk-3 的演化結果及各演化耗時

表 4.22: 實驗 Trunk-3 - 共 10 回合的最佳個體之標準化加權適應值

回合	指標	適應值	權重
	總適應值		
1	阻擋點	0.8165	1
	巡邏點	0.4730	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.7894		
2	阻擋點	0.7071	1
	巡邏點	0.4915	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.6986		
3	阻擋點	0.7071	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.7071		
4	阻擋點	0.7071	1
	巡邏點	0.4787	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.6858		
5	阻擋點	1.0000	1
	巡邏點	0.4818	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.9818		
回合	指標	適應值	權重
	總適應值		
6	阻擋點	0.8165	1
	巡邏點	0.4841	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.8006		
7	阻擋點	1.0000	1
	巡邏點	0.4966	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.9966		
8	阻擋點	0.8165	1
	巡邏點	0.4819	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	3.7984		
9	阻擋點	1.0000	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	4.0000		
10	阻擋點	1.0000	1
	巡邏點	0.5000	0.5
	守衛點	0.0000	-1
	物件數量	2.5000	2.5
	4.0000		

4.3.4 完整關卡演化結果

第 5 章 結論與後續工作

5.1 貢獻與結論

在實驗結果的章節中，房間容器依照遊玩特徵指標進行演化的結果，即使透過抽象化（體素化）空間設計其遊玩特徵指標，精確度降低的情形下，其結果仍具有一定品質與遊戲體驗。且提出之適應值標準化方法能夠有效解決多目標最佳化問題 (Multi-objective optimization)，讓不同的適應值能相互均衡發展與演化。綜合所有方法所彙整的關卡自動生成工具 *Dungeon Generator*，提供了完整流程的關卡生成解決方案。

藉由 *Dungeon Generator* 高度抽象遊戲概念的遊戲開發輔助工具，主以視覺化其概念邏輯之結構，輔以高度語意化的標籤進行數值微調，大幅降低關卡設計師的設計成本，終以加速關卡的生成產量。且 *Dungeon Generator* 藉由有意義為核心目標進行關卡生成，消彌了市面上絕大多數關卡生成工具的隨機不確定性，便能夠控管玩家的遊玩體驗。



5.2 限制與未來研究方向

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未來遊戲開發上必有相當大的幫助。

此外，若能夠將遊玩的歷程加入節奏 (rhythm)，隨著遊戲進程調整遊戲的難易度，令前述適應性函數權重進行動態調整，方能夠帶來更加不同的遊戲體驗。更進一步地導入機器學習技術，讓程序學習良好的關卡配置，建立其數學模型以自動化規劃遊戲特徵的權重。

參 考 文 獻

- [1] 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.
- [2] 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.
- [3] J. Dormans *et al.*, *Engineering emergence: applied theory for game design*. Creative Commons, 2012.
- [4] 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.
- [5] A. Liapis, “Multi-segment evolution of dungeon game levels,” 2017.
- [6] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
- [7] “The binding of isaac: Rebirth wiki - rooms.” <http://bindingofisaacrebirth.gamepedia.com/Roms>. Accessed: 2017-07-23.
- [8] A. Lipowski and D. Lipowska, “Roulette-wheel selection via stochastic acceptance,” *Physica A: Statistical Mechanics and its Applications*, vol. 391, no. 6, pp. 2193–2196, 2012.
- [9] A. Baldwin and J. Holmberg, “Mixed-initiative procedural generation of dungeons using game design patterns,” 2017.
- [10] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
- [11] A. Konak, D. W. Coit, and A. E. Smith, “Multi-objective optimization using genetic algorithms: A tutorial,” *Reliability Engineering & System Safety*, vol. 91, no. 9, pp. 992–1007, 2006.

