Level Design as Model Transformation: A Strategy for Automated Content Generation

摘要

本文將遊戲中，設計關卡的過程作為一系列的模型轉換。利用特定的設計原理和應用來進行轉換，例如使用*鎖與鑰匙*將線性的任務轉換成有分支的空間。這顯示只要透過使用**改寫系統 (Rewrite systems)**，這些轉換能夠被形式化與自動化。自動化的過程是高度可控性的：在設計關卡時，人與電腦能夠完美地透過**人機交互主導 (mixed-initiative)** 來合作，進行關卡生成。最後提出一個實作這些構想的試驗原型。

目錄

其它

原文作者：

- Joris Dormans (j.dormans@hva.nl)

譯者：

- Ze-Hao Wang (salmon.zh.tw@gmail.com)
- Pin-Lin Chen (ally53628@gmail.com)
- Yu-Che Cheng (z3010019@gmail.com)

審稿者：

- Kevin Lai (s900127127@gmail.com)
- Li-An Lin (jenny84311@gmail.com)

1. 介紹 (Introduction)

大多數的關卡設計手冊，相當落實地描述設計過程。它們為讀者提供一些理論工具來描述關卡，這些工具通常快速聚焦於應用程式中的任務。雖然許多使用流程圖 [15, 5]、問題分層 [7, 1] 和關卡布局 [5, 1, 16] 這些形式，來表達抽象的描述，但這些都沒有被開發成標準工具，讓設計師用合適的抽象方式，來思考或傳達他們的研究。這些的描述和理論，在許多方面彼此矛盾。而這些都沒有提出一個清楚且簡潔的理論，來實際說明什麼是關卡，以及關卡設計的質量是來自哪裡。

在本文中，我提出將關卡設計視為一系列的**模型轉換 (model transformations)** 的議題：關卡設計師產生一系列不同的模型，慢慢地逼近完整的關卡。即使大多數的關卡設計師不認為自己的產物是模型，我還是認為他們的產物大多是：關卡佈局的初稿，或一個只專注於特定方面的分鏡。這些模型互相有關：第一個模型將以某些方式影響，甚至主宰第二個模型的結構。討論的目標有兩個：它可以作為一種讓設計關卡（部份的）自動化的策略，也使設計過程形式化；**模型轉換 (model transformations)** 允許設計師以結構化或抽象化的方式，去推斷關卡設計。

模型轉換是從**模型驅動工程 (model driven engineering)** 的作法中，或計算機科學中的**模型驅動架構 (model driven architecture)** 所取得的概念。模型驅動工程將創建軟體，描述為一系列模型轉換的過程，例如，一個商業的模型轉換成軟體架構，而該軟體架構又可以轉換成軟體代碼。模型驅動工程是一種被設計來應對複雜性如企業級軟體的方法。它很大的程度上取決於形式化的概念框架，通過不同的模型表示，可以用於設計系統和系統架構之間的連結。它也在軟體自動生成中，擔任很重要的角色。其中一個主要原因是，它讓工程師減少了許多繁瑣、手動的工作，並提升程式設計的工作到更高層次的抽象概念，最高層次的抽象概念是由程式設計師的創造力和聰明才智。通過模型驅動工程，可以提高軟體生產的質量和效率 [4]。模型驅動工程可以在許多不同的模型上作業，其中一些領域是特定的，而其他則是通用的。非常強烈推薦使用 **UML** 作為標準建模語言，進而不受平台和工具影響 [17]。

這不是第一次將模型驅動方法用於遊戲開發。在一短篇論文— Emanuel Montero Reyno and Jos´e ´A Cars´ı Cubel [13] 中，探討了標準統一建模語言（UML）之技術和工具的使用，也探討了遊戲原型大部分自動化的創建。他們認為 UML 的方法比起遊戲設計師，更適合軟體工程師。同一作者在後續的論文中，擬定了一個針對遊戲規範的**平台獨立模型 (Platform-Independent Model)** 語言 [14]。儘管他們將遊戲結構圖和規則集合圖添加到 UML 所提供的模型集，以便處理遊戲中較細節的領域，但這種建模語言仍重度仰賴 UML 的使用。他們的方法和這裡所採用的方法，主要區別是，他們的方法僅只於遊戲的自動生成和建立原型，然而本篇則使用模型轉換去形式化設計過程本身。因此，我所使用的模型並非基於 UML，而是更接近設計師常使用的模型；同時是更接近設計領域的模型，而非軟體工程的模型。

2. 任務與空間 (Mission and Space)

模型驅動在有適合的模型可用時，便可應用於關卡設計。在先前的研究中，我對關卡任務和關卡空間提出了兩個不同的模型，使關卡方便以不同的步驟自動化生成 [9]。關卡任務代表玩家需要按照任務流程，來依序挑戰才能完成該關卡。一個關卡的空間由它的地理布局組成，或者由地圖呈現，或者由非常像地圖的節點網絡所組成。作者認為*一個完整的關卡需要包含空間和任務*；完整的關卡需要有一個特定的空間布局，和一系列需要在此空間執行的任務。

關卡設計的實作在理論上不該將任務從空間中分離開。因此，由幾位作者所提出的關卡設計布局的類型是相當矛盾的，有時令人困惑。由 Marie-Laure Ryan 所提出，最完整的**類型學 (typologies)** 之一 [15]，她的重點更側重於互動敘事而不是遊戲，她的範疇大多與適合任務的**拓樸結構(topological con-structions)** 緊密相關：「**故事樹 (story trees)**」、「**編織情節 (braided plots)**」、「**直接的網絡 (directed networks)**」、「**支線的引導 (vectors with side-branches)**」。但她也提到些明顯是空間結構的「迷宮」和「故事世界」。Ernest Adams 和 Andrew Rollings 的類別首先提到空間結構，但他們的範疇是利用不同的策略去促使玩家經由關卡來進步，在這種情況下使他們更能接近任務。

任務和空間的混亂常常導致關卡設計者採取簡單，但是有效的策略：使任務與空間**同構 (isomorphic)**。雖然這對於某些遊戲是非常適合的，特別是對於一個具有線性的關卡設計，當然，這並不是唯一的選擇。把空間與關卡分離開讓遊戲有更豐富的關卡設計策略。舉例來說，遊戲可能會為了不同的任務而重複使用相同的空間，像是在 *System Shock II（網路奇兵2）*中，玩家多次穿梭宇宙飛船的相同區域。在 *System Shock II* 這款遊戲中充分表現出相同的空間可以容納不同的任務（假設單個任務結構彼此不太相似）。以這種重複使用空間的方式是非常實惠的：開發者不必為遊戲的每個任務而創造新的空間。這種做法也有玩家在遊玩時的好處，舉例來說，玩家可以利用先前地圖的知識，以增加玩家的**主體支配感 (sence of agency)** 與遊玩深度。我將在下面論證，對於動作冒險遊戲，從空間中分離任務也是一個有用的策略，因為它允許我們去設計或者是生成一個線性較低的關卡，並強調玩家成長的經驗。

Figure 1: Level design as series of model transformations. The steps here correspond with the steps in of the generation process investigated in this paper in detail.

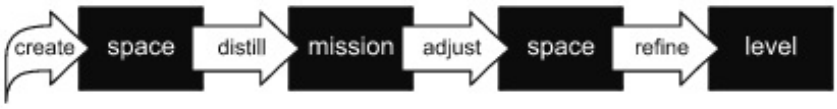

Figure 2: An alternative series of transformations. This paper includes some suggestions on how such a process might be setup, but does not go into details.

在 [9] 先前提到的自動化關卡生成過程，我提出了一種自動關卡設計的方法，藉由產生一個任務，然後使用這個任務去產生一個適合此任務的空間。可以想像，關卡設計者也能使用這種方法。他們可能首先通過生成任務的列表來創建任務，玩家必須執行這些任務才能完成關卡，接下來他們把任務轉變為空間，藉由重新安排任務在關卡的地圖上。設計者接著在地圖添加細節內容，直到該地圖充分塞滿任務要素並以做為遊戲關卡（參見圖一）。

如上述之任務與空間，每一個都呈現了關卡的不同面貌；每個要素都凸顯了同一關卡的不同結構品質。**任務圖 (mission graph)** 注重於任務與玩家的相互關係，而**空間圖 (space graph)** 代表了關卡的空間結構。通常，後者是更加複雜與詳盡；若有足夠的內容可以假設任務被嵌在空間圖之中，而不是相反過來。同樣的原因，當設計一個關卡時，首先設計任務通常比較簡單，然後設計一個空間來容納它。或者，設計者可以先設計一個空間，然後設計一個與空間相匹配的任務，或者在添加細節內容之前進行一些調整，使匹配任務更為便利（參見圖2）。這種方法更適合讓空間按照一些邏輯的架構來生成關卡：首先關卡可能是一個礦井，提供所有玩家期望從這類環境得到的所有元素，並可建構出一個符合該任務的環境。這樣一來，同一個空間也可以承載多個任務，如在 *System Shock II* 中的情況，其中玩家穿過太空船的甲板，並且在遊戲的後期階段返回先前所探索過的甲板。本篇論文主要專注在第一項，更簡單的策略。

3. 改寫系統 (Rewrite Systems)

一個模型能夠透過**改寫系統 (Rewrite Systems)** 來被轉換成另一模型。改寫系統由具有**左側**與**右側**的**規則 (rules)** 所組成。這些規則指定一符號集（左側）能夠被另一符號集（右側）所取代。此類操作類似於**形式語法 (Formal Grammars)** 中採用的規則。形式語法（或稱**生成語法 (Generative Grammars)**）起源於語言學，被作為述在自然語言中，由出現的詞組、片語集合的一種模型 [6]。形式語法通常對字串進行操作，但並不全然。**圖狀語法 (Graph Grammars)** 是形式語法的一種特殊格式，由多個節點與邊所組成。**圖 (graph)** 比字串更利於表達任務結構，且也能夠用來表示空間。在圖狀語法中，一個或多個節點與其相連邊會被取代成另一組由節點與邊所構成的新結構 [12]。圖三與圖四描述了取代的過程。經過特定的規則所描述，選擇一組即將被替換的節點之後，根據規則左側對所選擇的節點進行編號（圖四的第 2 步驟）。下一步，將所有被選擇節點之間的邊全部移除（第 3 步驟）。有編號的節點接著會被取代成**規則右側**的等價節點（相同編號之節點）（第 4 步驟）。然後將所有**規則右側**沒有**規則左側**等價的節點添加至圖中（第 5 步驟）。最後，連接新節點的邊依**規則右側**放入圖中（第 6 步驟），以及移除編號（第 7 步驟）。請注意，圖狀語法允許刪除現有的節點，但此操作在本文中不會被採用。

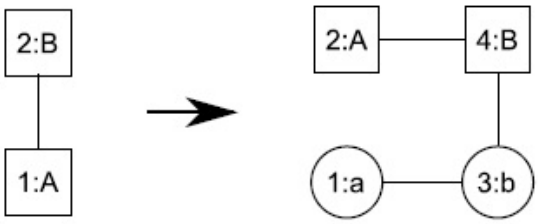

Figure 3: A graph grammar rule. Square nodes denote nonterminal symbols and circular nodes denote terminal symbols.

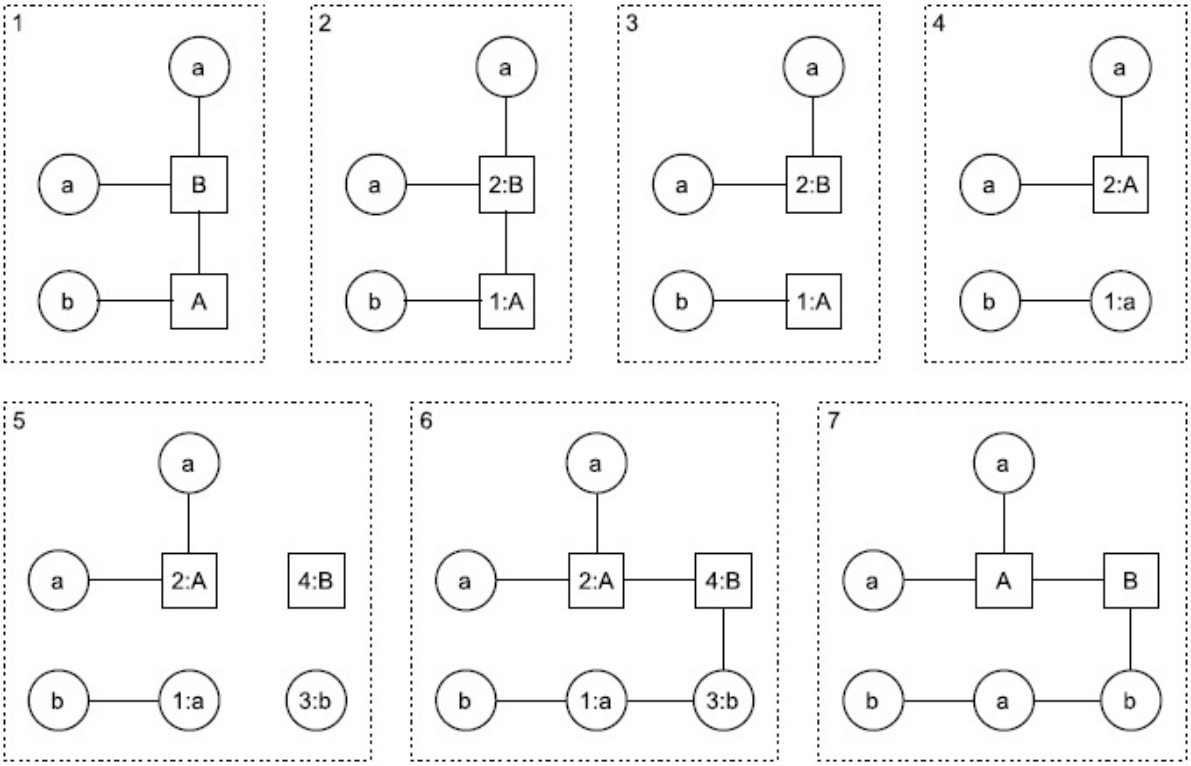

Figure 4: The process of applying the rule depicted in Figure 3 to a graph.

形式語法與改寫系統的差異在於後者能夠以一組符號作為起始點，且終端與非終端符號之間並沒有明顯的區隔。這意味著改寫系統**終止 (terminate)** 的方式與形式語法不同，任一**轉換(transformation)**將會產生有意義的模型 [10]。

改寫系統只能對符合形式語法的模型進行操作。在此特殊情形下，圖狀改寫系統可以從代表著任務的圖開始，並將其轉換為空間。改寫系統的規則必須以這樣的方式建構，使得輸出的模型不會與目標模型的語法互斥。在圖五中描繪了任務與空間模型、相互關聯的語法以及改寫系統。

改寫系統與形式語法是不同的，因為前者並不定義語言或是模型。然而，它們可以編改設計原則：改寫系統允許設計者對模型進行操作，讓模型轉換為另一個模型。當實行自動轉換時，改寫系統相當嚴格，僅允許操作設定好的規則，其餘概不接受。現實上的設計師更加靈巧，但他們也是遵守著某些限制。如果目標是創建出一可以通關的關卡，沒有一個設計師會在鎖的後方放上可以開啟這道鎖的鑰匙，因為這樣會導致**死結 (deadlock)** 的問題衍生。

使用改寫系統的優點便是防止會產生**死結**的操作，這需要根據某些限制來建構改寫系統，並且套用的轉換必須符合改寫系統的規則的。對於人類設計師來說，這也許不是最佳、最容易的作法，但可以很容易地達到自動化。可以開發一套設計關卡的軟體工具，基於改寫系統來生成任務、空間模型，來完成所有可能操作的工具。這類工具的附加優點是能讓設計師快速、有效的產生不同且正確關卡。此外，可以想像對於特定類型的遊戲時，全部的關卡可以透過此方式來自動化設計。

圖五建議將改寫規則套用在**任務圖 (a graph representing a mission)** 來產生**空間圖 (a graph representing a space)**，不過這並非必備條件。正如先前提及的，任務轉換到空間的過程涉及了許多小型的轉換步驟，而每一個轉換步驟由不同的改寫系統所控制。第一道改寫系統建立了數個任務，第二道改寫系統可以添加一些**依賴關係 (dependencies)** 或者是確保任務處於一有趣的順序。下一道改寫系統可以添加**鎖與鑰匙**來建立出非線性、更像空間的任務結構，而另一道改寫系統可以添加額外任務與獎勵。從任務逐漸轉換成空間形成了一種強烈的對比，任務與空間只不過是對遊戲關卡有用的觀點之一，轉換的過程中存在許多中間觀點。然而，這些觀點有著各自的特徵，有經驗的設計師利用這些特徵去蕪存菁後，能創造出令人嘆為觀止的遊戲體驗。

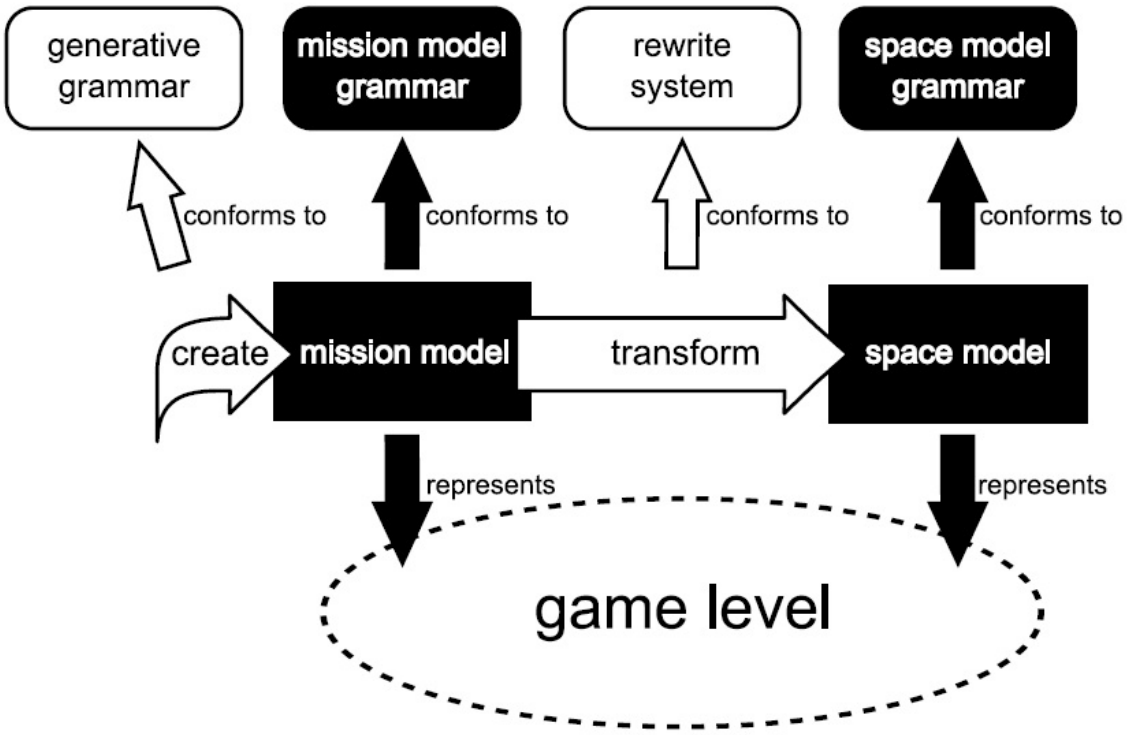

Figure 5: Level design as a model transformation.

4. 轉換範例：鎖與鑰匙 (Example Transformation: Locks and Keys)

改寫系統 (Rewrite systems) 可以被用於編制關卡設計原則。舉例來說，可以設計一個改寫系統，實作出在動作冒險遊戲中常見且典型的鎖與鑰匙結構（參見 [2]）。鎖與鑰匙是使用改寫系統的技術的重要例證，不僅是因為鎖與鑰匙是動作冒險遊戲常見的機制，而是因為它們對一個特定的關卡任務如何轉換成一個像關卡空間的結構，起著重要的作用。

鎖與鑰匙可以是字面上的鎖跟鑰匙，但他們相當普遍被用來偽裝成不同的項目。舉例來說，*The Legend of Zelda: Twilight Princess* 的玩家發現在的第一個地下城關卡中發現的道具「疾風迴旋鏢」，可作為武器跟鑰匙等不同的使用方式。它具有憑借風力操作開關的能力。遊戲的主角 Link 需要操作這些開關，控制幾個轉彎的橋樑讓他可以前往新的區域。為了得到可以解鎖最後魔王房間的鑰匙，主角必須使用迴旋鏢依正確的順序觸發四個開關。同時，迴旋鏢可以用來收集遠距離的道具 (這些道具可以用來獲得小物品和其它的道具），並也可以作為武器使用。這允許設計者將任務的後半部分的元素（在守護迴旋鏢的小魔王被擊敗之後）放置在用於任務的前半部分的相同空間中。這意味著玩家最初會碰到他們無法克服的障礙，直到他們找到正確的「鑰匙」。通常最好在鑰匙之前有這樣的鎖，有三個原因:

1. 當鑰匙一開始被玩家找到，玩家將被迫收集他們遇到的道具而不加區分，這造成了相當簡單的遊戲玩法。
2. 用別的東西像障礙物和物品作為鎖和鑰匙來呈現，如果玩家知道鎖是什麼將使玩家更容易識別鑰匙，玩家可以了解何處是他們該前進的道路；他們將會主動規劃如何返回鎖所在之處。
3. 當玩家可以通過他們以前不能通過的障礙，他們將體會到進步與成就感。

鎖與鑰匙可以讓設計師去做一個線性系列的任務，這本身就會產生一個同等的線性關卡，並將其轉換為一個分支結構（見圖六）。這個轉換僅能藉由兩種同型改寫規則來改寫（參見圖七中的規則 1 和 2）。這一步驟是重要的：任務的線性列表很容易凸顯出來，但是當直接映射到空間時，它不一定是一個有趣的關卡。鎖與鑰匙的分支作為一定程度的非線性。特別是當鎖需要多把鑰匙時，代表玩家可以依照任意順序去完成任務。

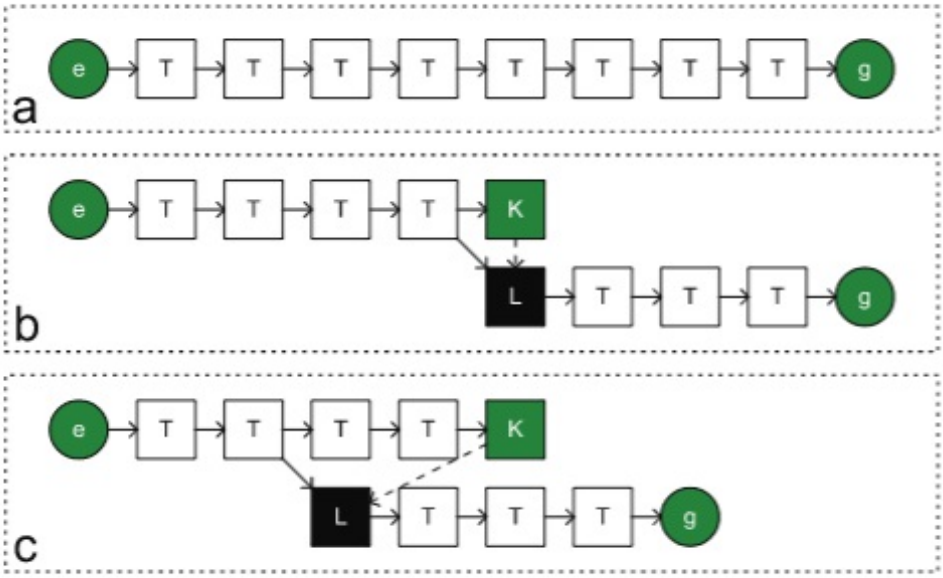

Figure 6: Addition of a lock and key transforms a linear mission (a) into a branching structure (b) in which the lock can be moved forward (c). In this model, e = entrance, g = goal, T = Task, K = Key and L = lock. The dashed line indicates which key unlocks what lock.

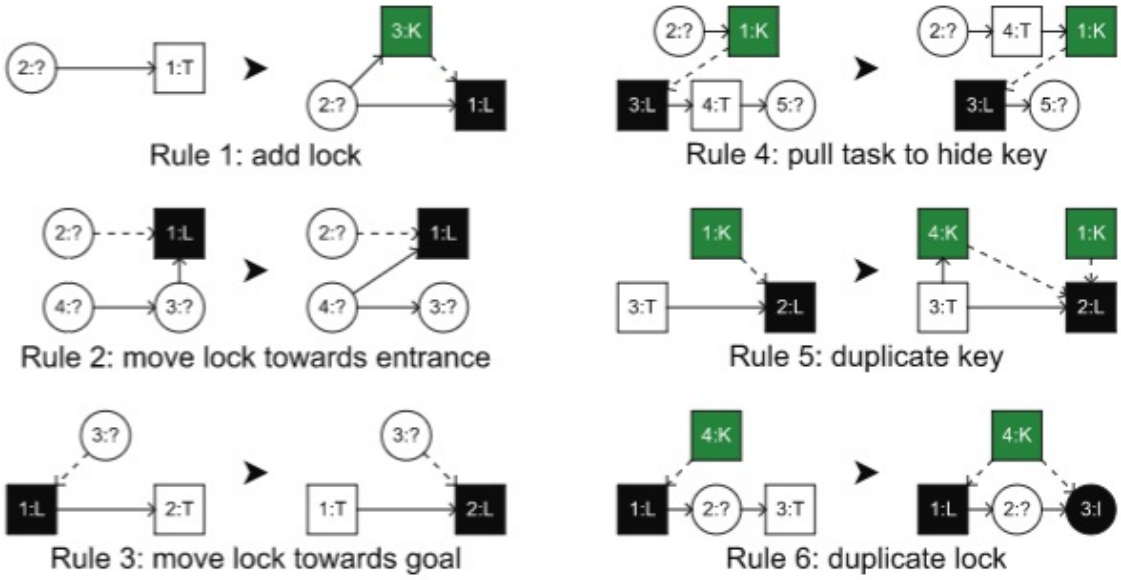

Figure 7: Rewrite rules governing the transformations enabled by the use of locks and keys. In these rules, the nodes marked with a question mark can be any node: the question mark acts as a wild card. Rule 1 might be translated into natural language as "if some mission element is followed by a task, consider turning the task into a lock and adding a key which is made available by the task". Rule 4 might read: "If a lock is followed by a task, that is followed by any other node, consider moving that task to precede the key that unlocks the lock".

有很多規則可以被加到基礎設定中，以生成更有趣的關卡。舉例來說，一個規則可以用來使一個鎖向目標移動（參見圖七中的規則 3），然而，這個規則打破了讓玩家在發現鑰匙之前，最好先讓他先碰到鎖的關卡設計知識。另一個規則是用來把鑰匙放在一系列的任務中，在遇到鎖之前。這將有效地隱藏鑰匙，確保玩家在找到鑰匙之前需要完成許多任務。其他規則包含使用多個鑰匙來做為一個鎖的開啟條件(規則 5)，或者是創造多次使用的鑰匙 (萬能鑰匙)（請參見規則 6）。注意在最後一個例子中，**額外鎖 (extra lock)** 是**終端鎖 (terminal lock)**，這意味這他不能依照規則 2 來移動。圖八顯示了些利用**改寫系統**來生成關卡結構的例子。

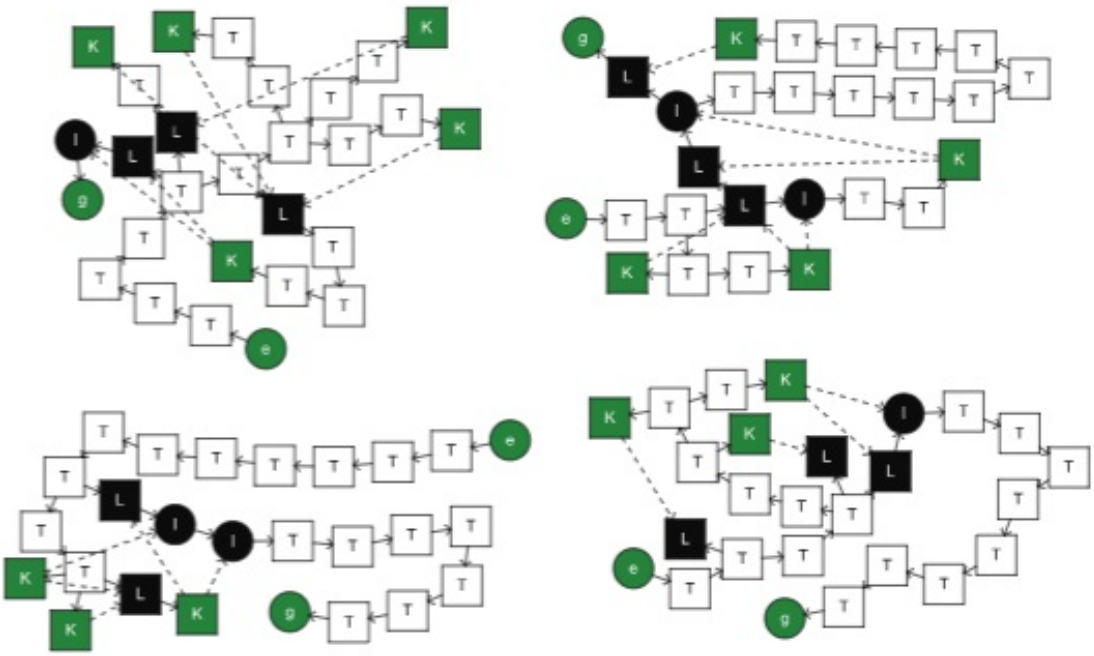

Figure 8: Sample levels generated by randomly applying the rules 1, 2, 4, 5 and 6 of figure 7 on a mission of twenty-one tasks.

使用改寫系統的技術是有高度可控性的。如果您將鎖和鑰匙組合視為單個任務，則這些規則都不會更改關卡中的任務數。這樣，關卡的大小由初始任務的長度決定。除此之外，這些規則還確保一個鎖始終被另一個元素跟隨。這可以通過調查規則來驗證：沒有一個規則允許去除在鎖之後的最後一個節點，且所有用創建鑰匙來當結束節點的分支，只能在其他地方進行。這意味著所有任務必須被完成才能通過這個關卡。這也是在圖七中規則 6 的第二個鎖，是無法被移動的終端節點的原因。如果可以移動，這將使完成關卡不再需要第一個門後的任務。

一旦生成了具有許多鎖和鑰匙的任務結構，就有許多策略可以建構空間去符合這個任務的所需環境。在先前的報告 [9]，我描述了一種使用**形狀語法 (shape grammars)** [20] 來定義空間部分的方法，這種空間方法與拼圖類似，被用來建構空間。雖然使用這種方法實作，但它難以產生任務空間來使多條路徑匯聚成同個目標。為了解決這個問題，我利用當表示二維圖形時的空間特性，使其可以容易地轉換成形狀。這種方法在 [3] 中概述。

5. 生成的機制（Generating Mechanics）

遊戲關卡生成的過程也可以擴展到遊戲機制，會需要使用**圖 (graph)** 的結構來表示遊戲機制。Machinations 框架 [8] 提供了這樣的框架，Machinations 流程圖已經被用於表示遊戲當中的內部經濟。他們模擬在**池 (pools)**（開放的圓形元素）上收集**資源 (resources)**（小型的彩色圓圈），池可能是被動或是互動式的。互動式的池以雙輪廓表示，並且可以透過某些玩家的動作來激活。箭頭表示資源如何在流程圖上流動，與**配特里網 (Petri-nets)** 的**令牌 (tokens)** 不同。虛線的箭頭表示池的狀態（池中的資源數量）如何影響其它地方的（稱作狀態連結）的量，或者在滿足某些條件（稱為**激活器 (activators)**）時某些元件是如何被激活的。狀態連結（state connection）含有*改變指示*的標記（「+」、「-」與「+2」），激活器含有*條件指示*的標記（「<3」，「>0」，「==3」）。其他的元件包含了，產生資源的**源 (sources)**（向上指的三角形），消耗資源的**渠 (drains)**（向下指的三角形），根據輸入與輸出的流量改變資源數量的 **轉換器 (converters)**（指向側旁的三角形），以及影響資源流向的**門 (gates)**（菱形）。如同池一樣，這些元件可以是被動（單輪廓）或是互動式（雙輪廓）。Machinations 可以用圖解表示，也可以受到相同類型的語法表示任務或是地形的空間。

改寫規則可以被用於編寫遊戲中發現到的**週期性結構 (recurrent construction)**，這些結構包含了一些代表性的遊戲目標 [11]。圖九中有許多的改寫規則，可以在遊戲中建構成多個遊戲目標。從這些初始結構的機制中不難看出，其能用更複雜的代替簡易的機制來拓展。可以在圖十中找到，描述前述轉換規則情境的範例。

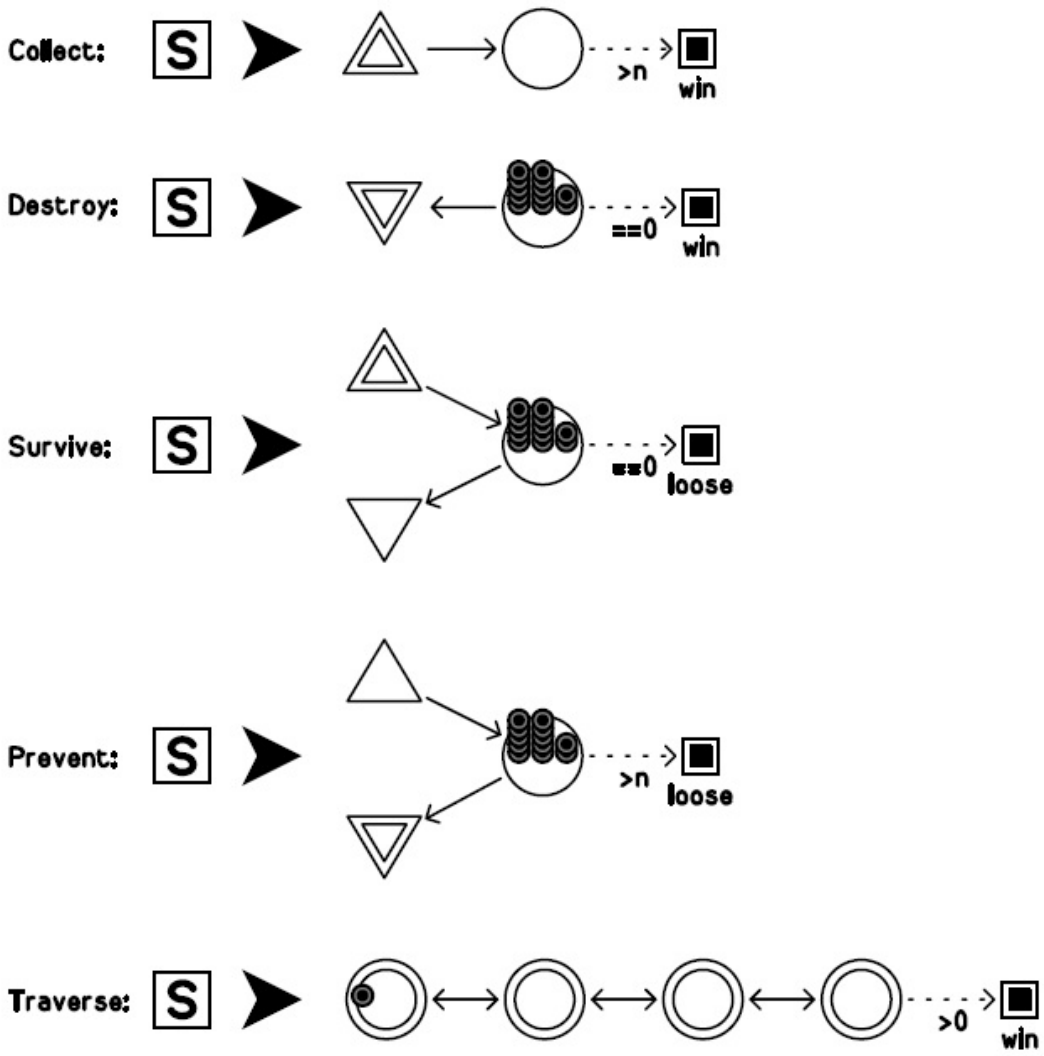

Figure 9: Transformation rules to create a goal from an arbitrary starting point (the non-terminal symbol "S").

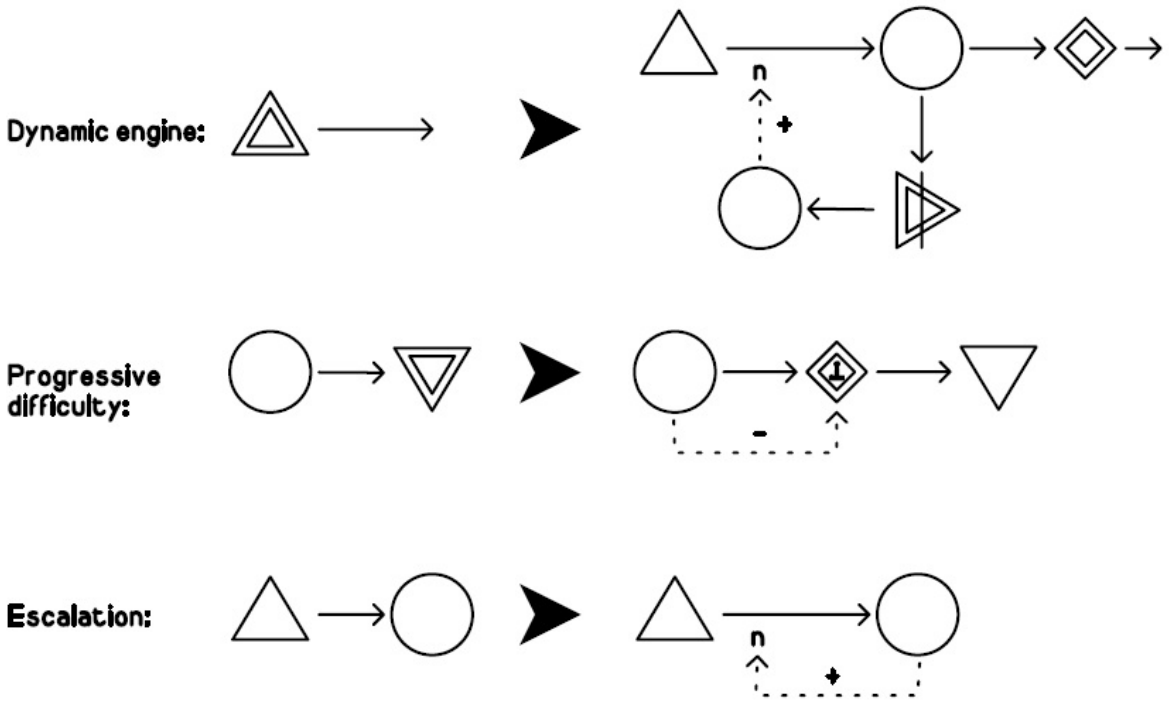

Figure 10: Rules to transform mechanics

機制 (mechanism) 與關卡之間的關係能以多種方式表示，舉例來說，能夠以在改寫系統中表達*鎖與鑰匙*的機制（參見圖七），這樣的機制並描述了在 Machinations 的關卡中的相同轉換規則（參見圖十一）。在基於 Machinations 的改寫系統中，變得能夠比**任務圖 (mission graphs)** 更方便地去包括更加複雜的機制。例如，一旦利用圖十一中的規則建立了*鎖與鑰匙*的機制，這些機制就可以被其他的*鎖與鑰匙*機制取代。圖十二中的規則提出幾個選項。同樣地，可以透過使用 Machinations 流程圖來更好地組成任務圖，用以表達其直接關聯的各個機制（參見圖十三）。

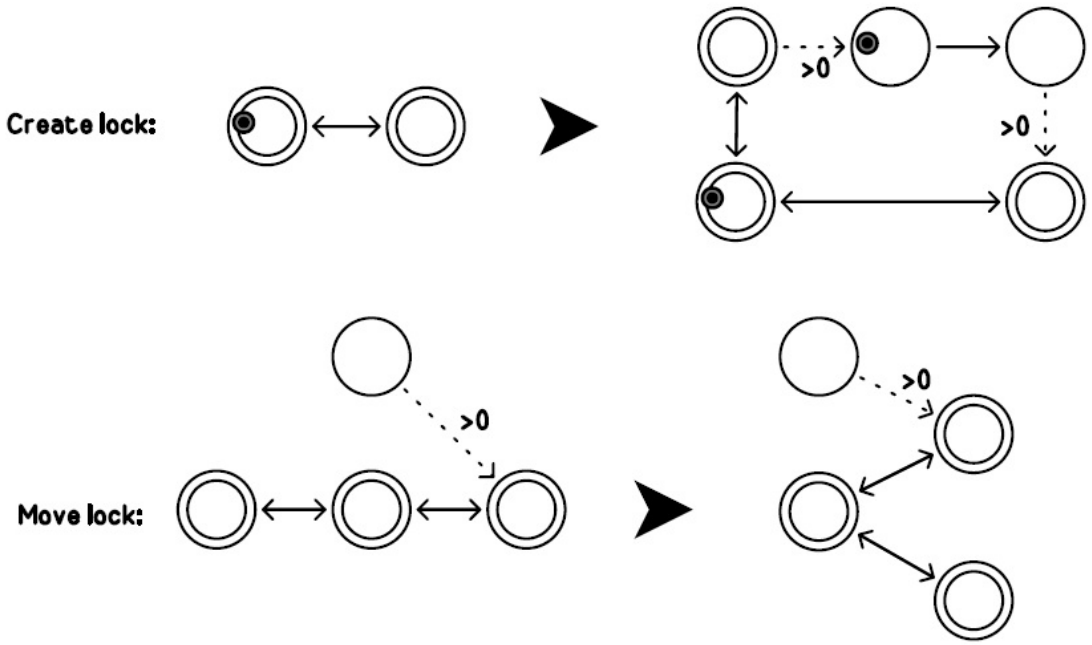

Figure 11: Lock and key transformation grammar expressed as Machinations diagrams. These rules correspond with rules 1 and 2 of figure 7.

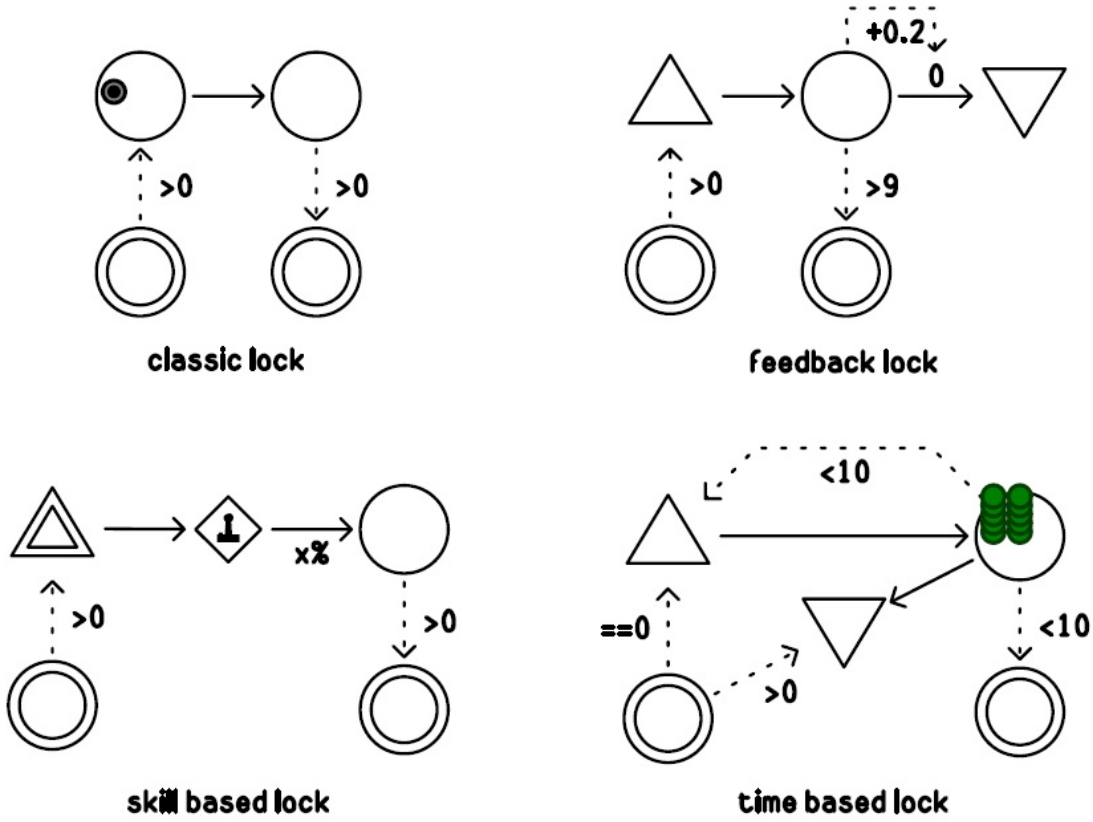

Figure 12: Optional lock mechanics. The classical lock can be replaced by any of the three other constructions in this figure, which in turn can be elaborated further.

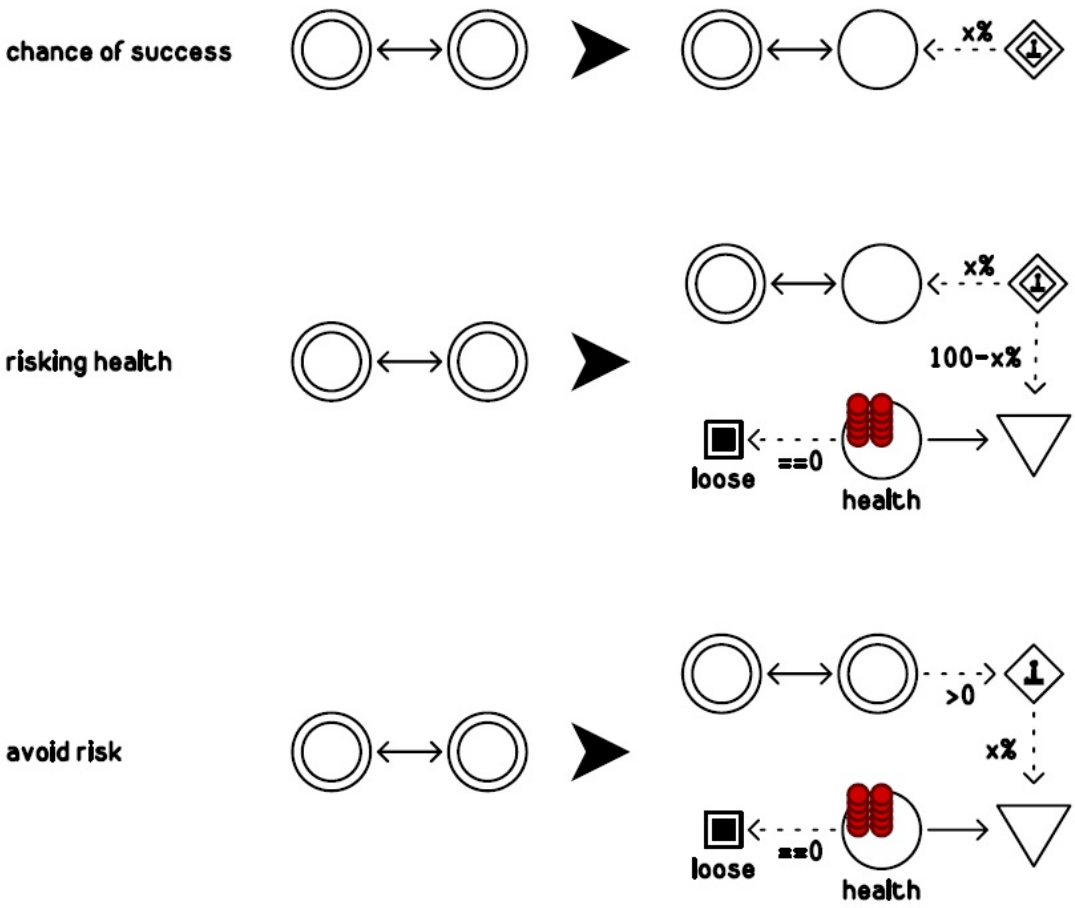

Figure 13: Rewrite rules that specify how tasks might be elaborated.

如同轉換被用於描述設計關卡的過程一樣，將會有許多種轉換的可能性。最直截了當的方式就是以任務作為開端，並以圖十一至圖十三所述之規則，來添加有趣的機制來完善。然而，更有趣的結果可能會是從機制出發，並且找到一種如何將這些機制轉換為有趣的任務之方法，尤其還找到一個方法去建立一個結構化學習曲線的任務。

解決方式可於生成第一個場景機制的過程中被發現。假設該過程以相當簡易的機制開始，像是圖九所呈現之，抑或從這些機制生成出一至二個精緻品、第一個關卡或是關卡中的第一個挑戰。可以從更進一步的轉換來生成後續的關卡（參見圖十四）。設計完機制的轉換歷史，可以作為關卡改進之基礎。此方法可以建構出相連的關卡，以及利用過往的**挑戰 (challenges)** 來準備給玩家當作未來的挑戰。在遊戲後期，可能會刪除部份的機制，以便採用新的機制來替代，藉此來創造遊玩的多變性。

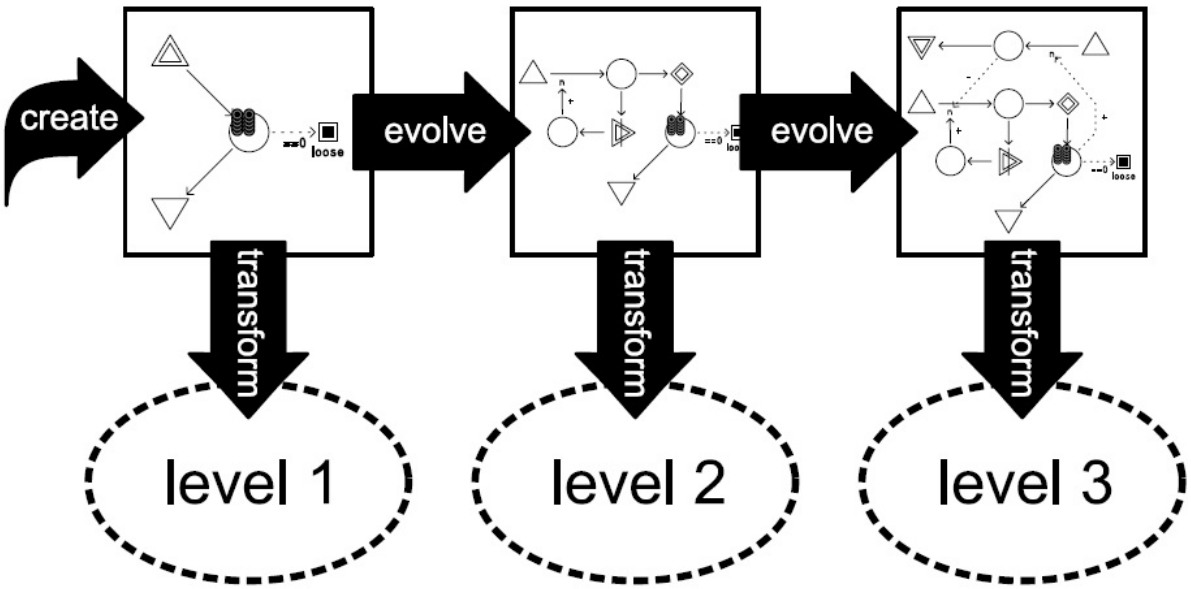

Figure 14: Steps in the generation of game mechanics could inform the generation of progressive levels.

6. 自動設計工具 (Automated design tools)

上述所討論的技術可用於自動化關卡設計工具。這些工具有幾種方法：一種是嘗試去建立一個從頭到尾完整生成關卡的工具，或是另一種是採用**人機交互主導 (mixed-initiative)** 方式去協助設計者的工具 [19]，亦見 [18]。雖然第一種方法本身是有趣的，但實際上完全使用生成來製作關卡的遊戲，相對較少。而我們對於專門協助設計者的工具很感興趣，而且已經有越來越多遊戲公司意識到這樣的工具，其可以有效提升他們員工的產出：可以讓關卡設計者更專注於他們工作內容中的創造力方面，並將大部分的手動工作讓電腦代替執行。像是有些遊戲有機會讓玩家在遊戲中成為共同創造者（例如 *Little Big Planet*）。

模型轉換和改寫系統對於人機交互主導方法很相配。它們提供設計者有許多的機會，在不同程度的抽象概念去控制關卡生成的過程。在最高層次的抽象概念中，設計者可以指定轉換的順序，為每個步驟選擇不同的改寫系統。實際上，這讓設計者能明確規範設計關卡時，是否把特定任務作為起點（如圖一所示），或是特定的空間是否有支配該關卡的設計（如圖二所示）。甚至有替代的模組可以來生成不同類型的空間：一個改寫系統可能產生一個 *dwarf fortress*，也可能產生一個 *orc lair*。附帶的轉換可能將 *orc lair* 換成 *dwarf fortress overrun by orcs* 等。為此目的，實驗原型允許設計師指定**配方 (recipe)**：一系列的規則被適用於合適的隨機節點。配方可以規範規則被使用的次數、工具隨機選擇的範圍或其可指定必須被使用的規則（只要該規則適用於合適的節點）。配方還可以指使工具選擇特定的改寫系統，或更改圖形的布局（參見圖十五）。

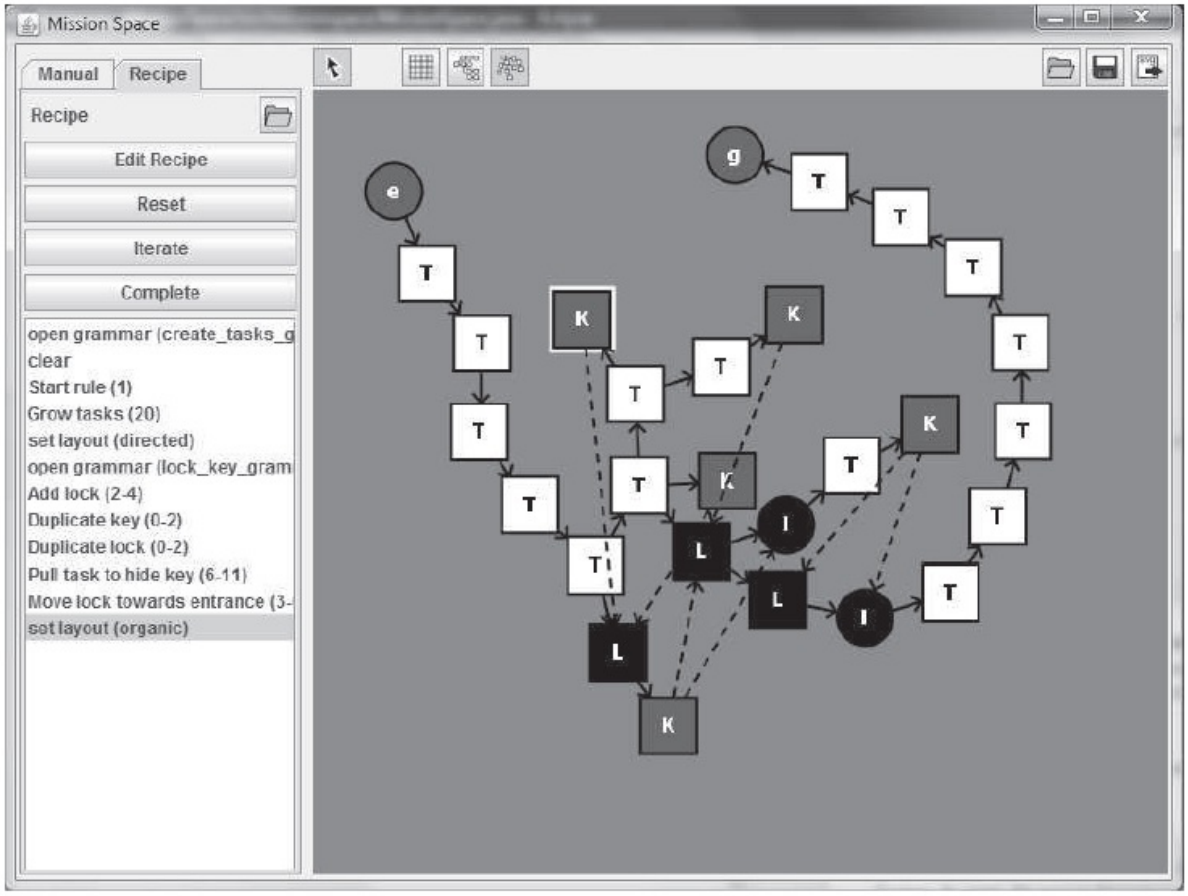

Figure 15: The experimental mission/space generator displaying a recipe (on the left).

在較低層次的抽象概念中，設計者也可能影響改寫規則的應用。在原型中，容許圖狀語法的架構和改寫系統，設計者可以在圖形中手動選擇節點，然後應用任何適用的改寫規則。當設計者沒有選擇節點時，這個工具就會找出哪些規則可以被適用於所有的節點，並提供給設計者去選擇。當設計者選擇要使用的規則時，會隨機選擇合適的節點（參見圖十六）。

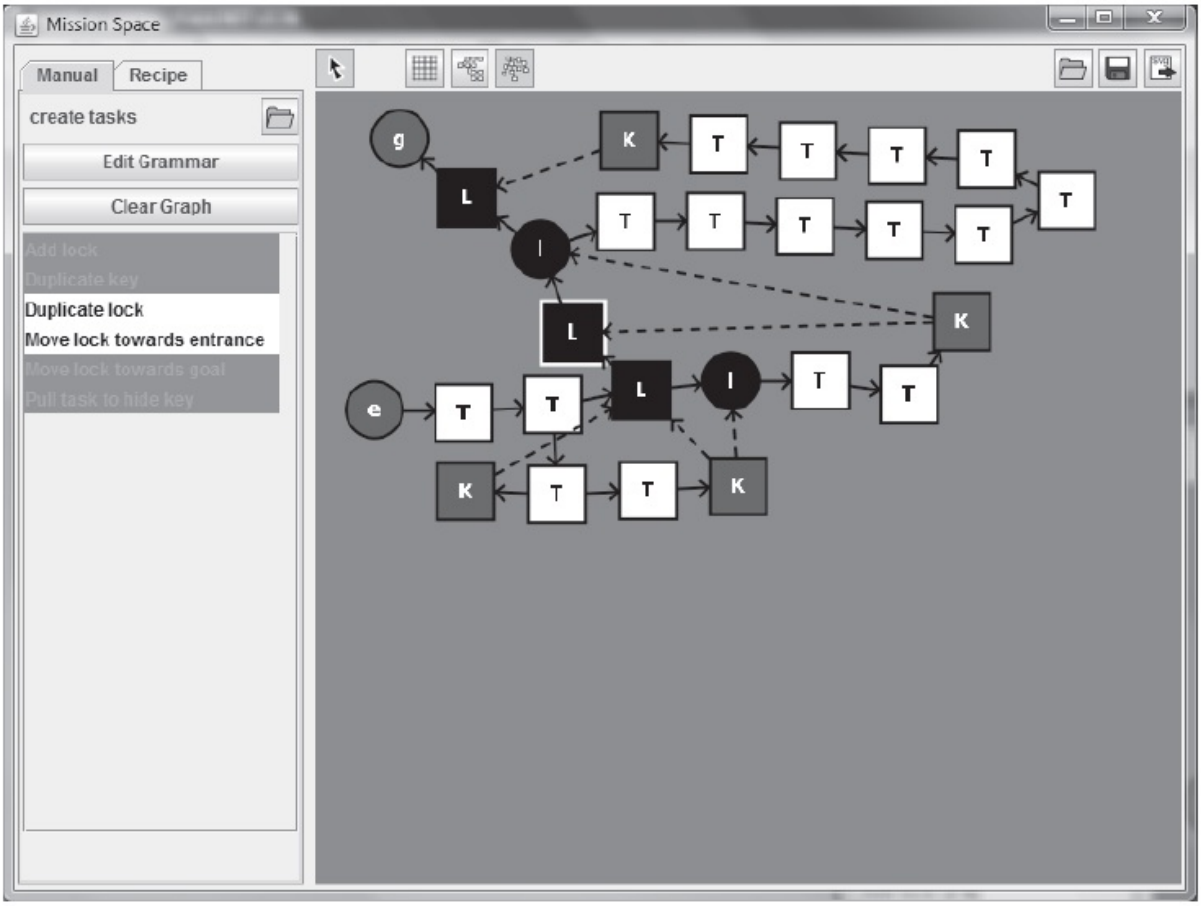

Figure 16: The experimental mission/space generator. The highlighted rules on the left indicate which rules are currently applicable to highlighted lock in the mission graph in the application's main view.

在這種方式，**任務圖 (mission graphs)** 會緩慢的被轉換成**空間圖 (space graphs)**，空間圖其中的特點是可以被轉換成地理地圖。該工具採用自動佈局系統來控制正在改變的**圖的表示式 (graph representations)**，但設計者可以通過拖移各個節點來手動改變佈局。目前，在 top-down 2D 動作冒險遊戲中，從關卡空間轉換到地圖的實作是非常明確的。以**形狀語法 (Shape Grammars)** 和形狀改寫系統來改進空間，也僅能在二維中實作，如果需要的話，同類型的語法可以在三維中執行。

7. 結論

本文研究關卡設計過程的部份自動化策略的**模型轉換 (model transformations)** 之使用。關卡設計可構築成一系列的模型轉換，讓我們能以改寫系統來正規化關卡設計的原則。此方法可應用於遊戲關卡的自動生成，並且讓關卡設計者能以更高階的抽象概念來達到他們的目的。藉由這樣的抽象化關卡，關卡設計者可以更專注在關卡的創造力方面。這增進了他們設計關卡的效率，以及減少了設計缺陷的機會。一系列的關卡設計原則，像是動作冒險遊戲中常見的*鎖與鑰匙*，其常用於說明如何被改寫系統建構出來，以及其與設計關卡的過程有何關聯。

模型轉換是彈性框架的一部分，設計者可以在空間生成之前，生成或設計任務，反之亦然。從剛開始到功能齊備、點綴完畢的關卡，可能會需要經歷相當多次的變換。將個別的轉換規模縮小使之有利，讓設計改寫系統更加容易，並創造出最大的靈活性。創造一個能讓關卡設計者，組織和控制大量的小型轉換的工具，同時提供了擴展關卡生成過程的機會，包含了遊戲機制的生成。以這種方式去涉及機制的優點，就是能夠更好地生成出遊玩性獨特的遊戲機制。

於本文中作為範例的轉換，皆建立了分支關卡，但是該技術還允許建構更加複雜的任務與關卡。舉例來說，可以從一具備兩個並行的任務串中，生成出一個有兩條選擇性叉路的關卡。使用**圖狀語法 (Graph Grammars)** 來表示任務與初版空間的優點，就是能夠相當自然地去整理整這種結構。

本文所討論的**圖形改寫系統 (graph rewrite system)** 幾乎是完善了，它能夠在設計過程途中，處理部分各種有用的轉換。這些轉換包含：困難的任務、添加鎖與鑰匙、添加額外格局與獎勵、生成遊戲機制來掌控玩家的進度、將機制轉換為任務圖、將任務圖轉換為遊戲空間的拓撲表示法，以及將該空間放入怪物、陷阱與解謎要素等，但不僅限於上述之。概略提及的**形狀語法 (Shape Grammars)**，它可以被用於建立形狀改寫規則來管理這些轉換，以及擴展生成、陳列空間的過程。

模型轉換和改寫系統所使用的是 **人機交互主導 (mixed-initiative)** 方式去生成內容，其中設計者將由電腦來輔佐他們的工作。模型轉換允許設計者控制關卡生成的過程中的種種不同細節，但同樣地能夠完全自動化這些關卡。作為本研究一部份的實驗工具，在實例中說明了它的運作方式。

然而，這項研究尚不完備。在撰寫本文之時，我仍在實作本文所描述的遊戲機制生成。此外，較少留意遊戲關卡與基本遊戲機制之間的關係，如遊戲世界中的移動、互動與衝突等，這些機制也會影響關卡的設計。為了提高遊戲的品質，關卡需要圍繞著這些機制去做設計或是生成。最後，從空間圖轉換到空間地圖目前是以一演算法來實作出，這個

方法特定於由上而下的二維空間。據我所知，沒有能夠將圖轉換為地圖，這樣現成的改寫規則存在。要研究如何用更通用的方法來實作前述的方式，投入更多的研究是有必要的。

8. 致謝

我要感謝 Remko Scha 和 Jacob Brunekreef 對這項研究的幫助。我非常感激阿姆斯特丹國際大學提供我探索這些知識的機會，且作為我的博士研究的一部分。最後，我還要感謝那些不具名的審稿者，他們的建議有助於改善這篇論文，如我在圖七的規則一中所「借來使用」的自然語言之翻譯。