Generating Emergent Physics for Action-Adventure Games

摘要

動作冒險遊戲通常集結關卡和含物理遊戲性的**漸進 (progression)**。為了生成這類型遊戲的內容，本文探討程序化技術如何擴展到生成關卡以外的領域，和生成**物理相互作用 (physical interactions)**。它建議一種形式化的圖形語言，來表示物理和遊戲實體間的因果關係網。利用轉換語法、模型驅動架構的原理和遊戲引擎的基本組合架構，人們認為物理圖表非常適合於生成自然的物理遊戲性。

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)
- Hao-Chen Wang (bachelor.whc@gmail.com)

審稿者：

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

1. 介紹 (Introduction)

目前遊戲的**程序化內容生成 (procedural content generation)**的研究，通常將遊戲內容解釋為故事、任務和關卡。大部分已發行且含有程序化內容的遊戲（像是 *Elite*, *Rogue*, *Diablo*, 或 *Civilization*），也注重生成關卡。在本文中，我想將遊戲內容的界線超出關卡設計之外，並探討遊戲機制如何生成。本文是我正在研究建立一個程序化生成的動作冒險遊戲的結果。雖然先前研究注重在生成這類型遊戲的任務和關卡，但在本文，我的目標是生成控制遊戲物理的機制。在動作冒險遊戲，新穎的機制對遊戲是很重要的一面，跟關卡設計一樣重要。成功呈現這類型的遊戲（如 *The Legend of Zelda*）將道具和能力與遊戲物理緊密結合，為玩家角色解鎖新的" 行動 "，和經常扮演" 鎖和鑰匙 "的機制；玩家通過被這些道具和能力控制的關卡而前進[2]。

在本文中，我將藉由在遊戲中不同實體間物理相互作用的生成，來更進一步我的研究。目標是讓鎖和鑰匙機制的創建，能夠在敵人、道具和能量上產生有趣的行為，但自己也能產生有趣的遊戲性。本文主要為理論性，概括了一般方法並提出了實施這些技術的方法；它沒有描述一個工作原型。但它提議了如何實行原型；它是基於當前的努力，來實行其中提出的想法。

2. 先前和相關工作 (Previous and Relacted Work)

程序化內容生成 (procedural content generation)是學術界和遊戲產業內增長的研究課題。在這個領域，遊戲機制的生成是相對較新且還未探索太多的研究。最近一項關於搜索基礎的技術用於程序化內容生成[25]的調查，列出了五個類別中的幾十個作品，影響了主要遊戲性和至少與選項內容一樣多的工作。在這些類別中，專注在規則和機制的類別只有五個作品，其中兩個專注在棋盤遊戲。不多說，遊戲機制的生成是一個巨大和複雜的議題，最好不要一次處理。Mark Nelson 和 Michael Mateas 建議將生成遊戲機制的問題分解為相互關聯的四個領域：**抽象遊戲機制 (abstract game mechanics)**、**具體遊戲表示 (concrete game representation)**、**主題內容 (thematic content)**和**控制對映 (control mapping)**[17]。這裡所提出的研究專注在第一個領域：抽象遊戲機制。本文的工作與 Adam Smith 和 Michael Mateas[22]採取的方法有很多共同之處。然而，Smith 和 Mateas 使用的程式碼在遊戲規則的基礎表示上，本文使用圖形符號來實現類似的目的。基於圖形表示遊戲機制的優點是，它們更親近非程式設計師，他們使用圖形語法會更容易生成[19]。從人機交互主導程序化內容生成的概念中吸取靈感的研究[23, 21]，知道易懂性是很重要的。該方法旨在使電腦和人類設計師在設計/生成過程期間合作，而不是在沒有來自設計師的輸入下，自動地生成完整的關卡或完整的遊戲。本文提出的研究遵循同樣的思路：目標不僅僅是生成遊戲機制，而是創建一個自動化的方法，幫助設計師快速且有效地創建遊戲機制。這種方法意味著，程序化內容生成本身不是一個目標，而是一種形式化和發展遊戲設計風格的手段。

本文基於我先前在關卡生成和遊戲機制形式化的工作[5,6]。特別是在 **Ludoscope** 的工作，關卡生成工具使用轉換圖形和空間語法來設計產生動作冒險的關卡[8]。這項工作建立在軟體工程中的**模型驅動工程 (model driven engineering)**的思想上，其中創建軟件的過程被分解為應用於表示領域或軟件的形式模型的一系列變換[4]。通過將設計動作冒險遊戲關卡的過程分解為一系列變換，並通過設計不同的變換語法來指定每個變換，我能夠創建一個有彈性的工具，生成各式各樣的關卡。在先前的論文中，我專注在設計動作冒險遊戲的過程中的兩個步驟：任務生成（玩家完成一個關卡所需執行的一系列任務）和容納任務的空間生成。用於每個步驟的形式模型是圖形。這意味著轉換由圖形語法描述[13]。

本文中採取的在動作冒險遊戲中物理相互作用的方法，與我先前的工作中採用的方法類似：不是以演算法為起點，而是先正式理解遊戲機制和建立在抽象概念的技術。在這種情況下，我將會看結構的正式描述和物理機制的質量，和他們如何有助於有趣的、自然發生的遊戲性。程序化技術將利用這些結構，以產生有趣的機制。

突現與漸進 (Emergence and Progression)

一個有趣的挑戰出自於嘗試生成動作冒險遊戲的內容，這些遊戲通常結合**漸進型遊戲 (games of progression)** 與**突現型遊戲 (games of emergence)** 的元素。這些分類由 Jesper Juul 首次提出 [14]，是創造遊戲玩法的兩種選擇方案。在漸進型遊戲中，遊戲玩法通常出於精心設計的關卡，向玩家呈現一系列愈來愈困難的挑戰。創造這類型遊戲需要創造出許多專門的內容。程序化內容生成針對已經成功應用在漸進型遊戲結構的關卡設計（舉例來說有 *Diablo* 與 *Torchlight*）。動作冒險遊戲使用關卡設計來建構遊戲體驗。同時，動作冒險遊戲也倚賴突現型的遊戲玩法。對戰機制經常採用特殊能力與增益道具的組合，讓其更近似於突現型遊戲。在動作冒險遊戲中，玩家構築個人策略與戰術，以克服特定的障礙。動作冒險遊戲往往被設計成便於探索的機制類型。

突現型遊戲倚賴於以少量機制來生成大量的可能性空間。術語「突現」源於複雜科學，複雜科學研究複雜、動態系統，包括但不限於遊戲。該領域的結果表明，複雜系統的許多結構特徵在塑造突現中，扮演著重要的角色。Stephen Wolfram [26] 建議以簡單的規則來管理多個狀態機之間的交互作用，便可以展現出令人驚訝的複雜行為。狀態機之間的連接數量以及狀態機的活動情形（它們多久改變狀態），是檢視其突現特性，不錯的指標。另一項研究中，Jochen Fromm [11] 將突現特性與系統內存在的**反饋循環 (feedback loops)** 相關連。以多個反饋循環來建構出更多的突現特性。

反饋循環與其遊戲設計間的關係，已被有經驗遊戲設計師給認可 [15, 20, 1]。在之前的著作中，著重在**經濟驅動 (economy driven)** 的遊戲（像是模擬、即時策略與桌上遊戲）的突現特性。我還得出下述結論，資源流動中的反饋循環是建構突現型遊戲的必備要素 [5, 7]。

為了成功地生成出動作冒險遊戲的機制，機制將由遊戲中的突現與漸進等特性，來適應至其結構性質。在這方面，鎖與鑰匙的機制發揮重要作用，因為鎖與鑰匙機制與漸進遊戲的結構息息相關，並且同時與遊戲的突現物理特性相互作用。

##4.鎖與鑰匙的機制 (LOCK AND KEY MECHANISMS)

鎖和鑰匙機制是任何動作冒險遊戲的核心特徵。鎖與鑰匙的機制被用於通過遊戲建構玩家的進展。一個典型的冒險遊戲有實際的鑰匙用於解鎖門，但也包括許多其他類似鑰匙的功能的物品。舉例來說，當需要打敗一個特定的敵人，一個特殊的魔法劍可以作為一個鑰匙，以便繼續進行遊戲。設計有趣的鎖和鑰匙的機制是設計冒險遊戲的一個重要方面。在許多發行遊戲的成功設計策略中通常圍繞著結合物品與增益道具作為物理模擬，戰鬥系統的鑰匙。這些鑰匙很少僅用於解鎖特定門口的唯一目的; 它們可作為武器或讓玩家可以以新的方式在環境中移動。舉例來說，炸彈可以用於敵人，但也可以用於清除被阻擋的通道。 為了讓玩家創造一個新的體驗，設計師總是在尋找新的和有趣的方式來創建這種類型的機制。

為了這個目的，集成到物理模擬是一個'關鍵特徵'，一般來說，最好通過設計一個系統用於最大化元素之間可能的相互作用的數量來實現集成，而不是通過引入許多不同（類型）的實體。這允許設計師設計需要玩家結合不同項目和/或動作的謎題。

鎖和鑰匙機制已經在**任務/構造(Mission/Frame framework)**中發揮了重要作用[9]。在特定的鎖和鑰匙機制中允許設計者在非線性分支結構（參見 圖1）中轉換線性任務集合。

這些分支結構更適合遊戲空間的生成。在這種情況下，通過執行**變換圖語法(transformation graph grammar)**來實現變換本身。生成整個關卡的過程可以被分解成各自由不同語法描述的一系列變換。然而，在這個先前的報告中，鎖和鑰匙仍然是抽象的結構，我沒有嘗試為他們生成更有趣的物理機制，或者在遊戲中給他們額外的目的。

通過在任務圖中添加描述鎖和鑰匙的任務的屬性，可以生成更詳細的機制和多樣的目的。圖2說明了這一點，屬性被表示為分別連接到鎖與鑰匙的六邊形。在這種情況下，該圖表示弓和箭如何能夠起作用用來打開鎖，需要在一定距離處對其施加傷害。這種情況可以通過創建門來實現，該門由需要被打擊的開關觸發，並且將開關放置在不能穿過但可以被射擊（例如水）的障礙物之後。圖3顯示出了滿足這些要求的空間圖。在該圖中，大圓圈表示關卡中的不同房間或地方，而黑色圓圈表示位於那些地方的物品或特徵。實線箭頭指示玩家如何在地點之間移動。虛線箭頭表示特定開關解鎖門，當實體的箭頭在水與開關之中，表示著開關是受水保護的。圖4表示可以生成構造的轉換規則。

類似的特性，像是造成火屬性傷害，獲取遠距離物品與抗火的能力可用於從一個和另一個之間分離鑰匙，也可以測試一個物品是否可能無意間做為一個為另一個鎖而設計的鑰匙。舉例來說，劍可以被看成一個具有傷害屬性的鑰匙，而弓作為具有損傷屬性以及遠距離屬性的鑰匙。這意味著，一個鎖可能藉由劍或弓造成傷害而打開。在這種情況下，生成一個關卡，令玩家找到弓之前的劍更加有意義。另一個有趣的案例是，將鎖設計成在熔岩後面，必需用弓來觸發。然而，如果同一關卡存在一個物品能讓玩家安全度過熔岩，則玩家就可以不需要弓而通過此區域。在這兩種情況下，轉換不會產生不想要的情況變得很重要。防止系統生成這些情況的最直觀的方法是指定一個**限制(constraints)**來識別他們並使用這些限制來消除在應用他們之前所生成的轉換。

##5.限制 (Constraints)

限制(Constraints)不是使用傳統變換語法組成部分。傳統的方法將需要以在特定建築還未產生的方法來建構語法。在許多情況下，這是可能的，但這也是困難的，它需要一個完整的理解與經驗對於轉換語法。遊戲的混合驅動程序內容生成使遊戲設計師成為重要的目標群體。我計畫創建自動化工具來幫助設計師。對於提出一些**限制(Constraints)**的設計師來說，如何設計出一個不會產生不必要的結構的語法是很直觀的。因為這個原因，我推薦使用**限制(Constraints)**來擴展 *Ludoscope* 中的轉換語法施行。

在 *Ludoscope* 中，**限制(Constraints)**與規則以相似的方法被定義，一個圖表被創建用來說明不應該發生的情況。舉例來說，在圖5中的**限制(Constraints)**指出用於特定門的鑰匙不能被放置在它解鎖的門後面。在這種情況下，交界箭頭上的星星表示在鎖與其鑰匙之間可能存在任何數量的連接; 此鑰匙需要不直接跟隨它的鎖。當語法在尋找適當的規則，像上述這樣的**限制(Constraints)**可以容易的檢查。 *Ludoscope* 在選擇要應用的規則之前已經編譯了所有是用規則的列表。如果規則的應用導致與**限制(Constraints)**符合的圖，則會刪除該列表中的規則。這使得選擇和應用規則的過程隨著過程強度增加而減慢。然而，符合子圖快速實行的存在並且可以在此情況下使用 (參見 Marlon Etheredge 在同一研討會上提交的論文[10])。此外，在編譯適合規則的列表的同時應用規則和尋找**限制(Constraints)**具有的優點是提供一些機會去運行這種會影響到規則的適用性的**啟發法(heuristics)**。例如它可以檢查新圖形的大小，並根據指定的目標大小調整規則選擇的可能性。，

以這種方法實現，**限制(Constraints)**的定義可以容易地擴展到包括有用的快捷方式。舉例來說在圖6中**限制(Constraints)**使用特殊命令，該命令詳細說明**限制(Constraints)**只能被應用在當相符被發現。其中節點1具有相同的屬性集合與節點2。

每個語法都詳細說明了**限制(Constraints)**，在*Ludoscope*中，語法的執行對應於多步生成過程中的單個變換。這意味著在每個步驟之後，生成的內容應該是連續的和一致的。 它還增加了後來步驟的靈活性，能夠打破適用於早期步驟的**限制(Constraints)**。特別是因為在用於內容生成的這種模型驅動架構中，後來的轉換越來越趨向於特定被期望的遊戲。舉例來說，在早期創建遊戲結構的通用轉換中不允許把鑰匙放在鎖後面，這是一個好主意，它等同於讓鑰匙被放置在鎖之後的語法。因為在這個特定的遊戲中，該鎖可以一開始被解鎖然後被鎖起來當玩家通過它時。

6. 物理圖表

鎖與鑰匙添加屬性以及使用**限制 (constraints)**，能夠在動作冒險風格的遊戲中，創建出更多種類的鎖與鑰匙。然而，這並不保證遊戲會生成突現的物理交互效果。為此，我們需要更加瞭解遊戲物理現象中的結構性質，進而產生突現與更直接的方式來模擬這些性質。

如上所述，突現可以歸因於遊戲設計中的結構性質。突現取決於活躍與相互關聯的元素們。對於由不同遊戲元素所產生、消耗的資源流動，所構成的遊戲內部經濟，反饋結構在突現型遊戲玩法中發揮了相當大的作用。當一元素於其它元素中產生變換，並隨著時間推移，在初始的元素中引發新的變化，這樣子的狀態改變將會形成反饋。事實證明，類似的觀點可以應用於遊戲的物理機制，我們需要在物理學允許的因果關係中尋找反饋循環，而非在資源的生產與消耗中找尋。這一點在 2012 年遊戲開發者大會近兩次的會談期間，被拿來獨立討論過。遊戲設計師 Randy Smith 談到了他最近關於 *Waking Mars* 的研究，闡述了在遊戲中建立長鏈型因果關係的重要性 [24]。例如，在 *Waking Mars* 中，玩家可能會在肥沃土壤上撒種子，使之發芽。植物產生新的種子落到地面，有可能會與遊戲中其它的**實體 (entities)** 是種子的天敵，包含了螃蟹的生物會將種子帶走並吃掉它。然而，玩家可能會嚇唬螃蟹，讓牠們放下種子等。曾在遊戲 *Osmosis* 工作的設計師暨研究員 Andy Nealen，建議將這些鏈結構築成循環的結構（從而創建出無限長的鏈結），為了由少數功能與遊戲機制建立出複雜性質 [16]。

相對簡單的觀點和允許互動的機制，讓遊戲中的突現物理特性，概略闡述了 2D 平台遊戲的成功與優雅。這些遊戲中幾個簡易互動（移動與跳躍）創造了許多可能與遊戲中其它元素的互動（在平台上跳躍、跳進平台以顯示隱藏的道具、跳在敵人身上或是走入他們之中）。通常互動會牽連另一個互動：在一個不穩定的平台上著陸，會讓該平台崩塌，以至於平台可能會壓死敵人等等。欲得知事件長鏈與其循環的存在與否，最簡單的方式就是藉由觀察有可能的交互作用之數量。突現物理特性出自相對少數元素間的種種交互作用。

因果關係的網絡很輕易以圖表來表達，因此很容易使用**圖狀語法 (graph grammars)** 來生成出來。因果關係網絡所需要的是以**圖 (graph)** 語言來建立，以表達出這些物理交互關係。對於 **Ludoscope**，我提出了一種圖語言，它具有三種類型的節點來表示**實體（原型節點）**、**條件（菱形節點）**與**動作（三角形）**。如圖七表示了用於簡易平台遊戲，其物理性質的網絡。在這種情形下，有一個稱為玩家的實體，響應了四種不同的動作：因重力產生的加速度、移動與跳躍動作、跳躍時的支持動作。在該圖表中有數個條件：當玩家與平台實體碰撞時才會啟動支持動作，移動動作僅在按下特定按鍵時有效，而跳躍動作僅於玩家被支持著且按下特定按鍵時啟動。

表示遊戲物理特性所需要**條件**的種類被限制。對於大多數的遊戲來說，檢查重疊或碰撞的條件，對於描述**實體**之間的物理交互關係有很大的幫助。其它許多**條件**可以從圖本身的語法衍生出：當一**動作**當前作用於一**實體**時，該**動作**與**實體**將可作為某一**條件**的輸入。表示玩家輸入的**條件**形成另一個高度限制的集合。

這種表達物理機制的方式，一個重要的原因是，一個**實體**可能包含另一個**實體**；**實體**可以作為其它**實體**的屬性。所有影響容器**實體**的**動作**，也會自動影響其包含的**實體**。便能夠以一種彈性的方法，來定義一款遊戲的物理機制。例如，圖八定義了劍的機制（左上方），簡單地描述了—如果劍與任一**實體**重疊，將會啟動該**實體**的損傷**動作**。圖八還定義了響應兩種動作「損傷」與「恢復」的生命實體（右上方），同時還能夠在這個容器上產生「被殺死」的動作。如果我們現在創建一個簡單的「敵人」實體，且這一實體包含了生命實體，敵人將會變成會被劍所傷以及致死。同時，我們可以透過啟動與其相關連的實體（如虛線箭頭所指），以及當啟動時所開啟的門（亦參見圖八）來定義對收到傷害而做出響應的開關。

如圖八所示，實體可以與其它實體分開來定義，或是於物理機制的完整圖表中（如圖七中的情形或多或少）。將實體分開定義，取決於動作的可能產生的隱含關係。劍將與它重疊的任何東西造成傷害，不過僅限於定義對傷害動作會進行反應的實體，或是含有做出相同反應的實體，將會對該動作進行響應。這讓我們相對簡單地在定義與完整的網絡間轉換。定義可以從一完整網絡中剪切，藉由選擇一實體並追蹤它的輸入及輸出，返回至最近的動作與實體，與它們之間的所有關係。同樣可以透過連接相同動作，從孤立的定義來建構出網絡。

因為交互作用的增加、因果關係的鏈長增長，讓遊戲中的物理引發更加動態的突現特性。當啟動動作實體以及響應來自與多實體共用的有限集合時達成。例如，愈多的實體響應了損傷動作，當玩家使用劍時，將會具備了更多的選擇。同時，響應與啟動了動作實體，令事件鏈提供了更多的機會。例如，現在劍只會啟動動作，門只會響應動作，開關、計時器和生命實體會響應暨啟動動作。一般來說，最好能夠以這樣的方式來生成或設計實體。

實作物理特性 (Implementing the physics)

利用**圖狀語法 (graph grammars)** 的**物理圖表 (physics diagrams)** 來定義出遊戲中的**實體 (entities)** 相當容易，不過要實作或產生與其相關的行為就相當地困難。我建議採用愈來愈受歡迎的**元件式架構 (component based architecture)** 遊戲引擎，來完成這項艱困的挑戰。這樣便能夠有效地處理各式各樣、不同行為的**遊戲物件 (game object)** [18]。相對於元件式架構的另一種選擇是繼承式架構，而後者較不適合處理不同類型的遊戲實體或遊戲物件的共用行為，而共用行為遵循了軟體工程的**組合模式 (composite pattern)** [12]。

元件式架構有兩項核心類別，分別為 Components 與 Composites（備註一），其中 Composites 涵蓋了 Components。不管怎樣，由於 Composites 是從 Components 繼承而來，所以其它的 Composites 也可以涵蓋之（參見圖九）。這讓彈性的巢狀架構緊密地契合上述之物理圖表，而物理圖表中的遊戲實體能夠被其它的實體所涵蓋。在元件式架構中，遊戲物件並不如其組織方式來定義，遊戲物件是透過了許多元件來創建，而這些元件定義了遊戲物件的行為，但很少去定義元件本身的行為。

> 備註一：雖然更嚴謹的實作模式會是有三種類型，分別為：Component、Composite 與 Leaf，其中 Component 是**抽象類別 (abstract class)**。

在元件式架構中，個別的元件間不會直接進行通信，而是透過訊息系統來通信。在任一元件都能夠發送與接收任何自定義訊息的情形下，透過 handleMessage 函數來實現該功能。Composite 類別覆寫了 handleMessage 函數，為了也能夠將所有訊息傳遞給它的元件。這樣的方式良好地將物理圖表中的動作給聯繫在一塊。

為了在元件式遊戲引擎中產生物理特性，能夠以兩種不同的方法達成。首先是第一個策略，遊戲物件可以由特定遊戲之中，其預先設計好的元件所構成。如果在這樣的情形下，您想要實體 A 與另一個實體 B 進行互動，您可以在實體 B 添加能夠響應實體 A 啟動的元件，反之亦然。圖十說明了**轉換語法 (transformational grammars)** 應用於此的情況，會造成損傷的實體及另一實體中，於後者添加了生命元件（包含了圖八中定義的所有行為）。此規則使用虛線箭頭來表示未指定的關係。注意，本規則還強制實體 B 必須響應死亡的動作。

第二個策略對遊戲實體或使用了**生成語法 (generative grammar)** 的元件生成定義，然後生成能夠實現該行為的程式碼。例如圖八中的生命實體，能夠很容易地轉換為以下的程式碼：

```
public class Health implements Component
{
  private hp:int = 3;

  @Override
  public void handleMessage(message:String)
  {
    if (message.equals("damage")) hp--;
    if (message.equals("heal")) hp++;
    super.handleMessage(message);
  }

  @Override
  public void update() //called every frame
  {
    if (hp==0) parent.handleMessage("kill");
    super.update();
  }
}
```

採用策略二，首先能夠產生物理網絡，並從中萃取出元件。這樣便可利用預期數量的交互作用與**反饋循環 (feedback loops)** 來產生出物理網絡。

結論

Ludoscope 是用以支援模型驅動方法的**程序化內容生成 (procedural content generation)** 軟體。該軟體包含**物理圖表 (physics diagrams)** 與**任務圖 (mission graphs)** 的擴充，具備僅有火光下可見鎖鑰組合的特徵。藉由圖形語言設計與遊戲設計的子領域，有可能將設計過程的每個設計步驟作為各種語言的圖形轉換。此外，圖形轉換可以適用在不同的模型中。例如，**生成語法 (generative grammars)** 與**轉換語法 (transformation grammars)** 可以生成物理圖表，額外的語法與任務結構同樣也適合運用在物理機制。這些設計過程不需嚴格地按部就班。物理特性可以在任務創造之前生成，反之亦然。

鎖與鑰匙的屬性在這一部份是有幫助的，它們在物理特性生成的過程中提供了 hook，或者在生成任務時負責關聯限制的規則。Ludoscope 也同樣支援**人機交互主導 (mixed-initiative)** 的方式到程序化內容生成，如設計師與電腦輪流地進行生成的程序。在這個案例中，設計者可以親自設計物理圖表，而電腦則產生對應物理的關卡。透過延伸人機交互主導的方法到遊戲物理特性，這套非常快速的雛型工具可以讓設計師以直覺的介面進行遊戲設計。本文所展示的物理圖表同樣也被全自動遊戲生的工具，像是 Ian Bogost 在遊戲開發者大會上，所提到的 Game-O-Matic [3]。在 Game-O-Matic 中，設計者給定一個內容地圖，當中包含了遊戲實體與其動詞之間的關聯性。透過圖形轉換，同樣可以產生物理圖表讓動詞能被轉譯成不同形式。透過**限制 (constraints)** 與經驗法則，這個工具能迅速地搜尋所有可能的實作與**反饋循環 (feedback loop)** 還有交互關係。

如今，本研究仍處於早期階段。物理圖表的轉換能在 Ludoscope 與文法中定義。而撰寫本論文的當下，尚未有將物理網絡實作進元件式架構的雛形。然而，在開發中的雛型與初步成果，結合我在程序化內容生成的經驗，相信這個方法在不久的未來會有有趣的結果。