ChatGPT

RimWorld 模組開發技術學習路徑圖與教材設計

學習階段總覽 (初階 → 進階 → 整合)

- **初階階段：**聚焦模組系統的基礎概念，包括**資料驅動設計**（以輕量格式描述遊戲內容）[1]與**Def 定義**（使用XML等格式定義遊戲物件）。本階段學習者將了解如何提供一個外部資料夾讓玩家放置模組內容，以及遊戲如何讀取這些資料[2]。同時介紹RimWorld中的**PatchOperation**機制，學習如何透過XML節點操作安全地修改遊戲定義，以避免模組衝突[3]。

- **進階階段：**涵蓋更高階的模組開發技術，包括**模組內容封裝與載入**（模擬 RimWorld 的ModContentPack資源/程式碼隔離）、**程式碼注入**（自動載入模組DLL並初始化模組類別）、**Harmony 核心程式碼修改**（Prefix/Postfix/Transpiler 三種patch技術)[4][5]、**模組設定檔與序列化**（使用ModSettings與Scribe保存設定），以及**模組相依性與載入順序**管理（處理模組間依賴關係與衝突警示）。

- **整合階段：**以上所有知識點將應用於一個**綜合實作專案**。學習者將開發一款簡化版的 Vampire Survivors 類型遊戲，並從零構建類似RimWorld的模組系統，支援動態新增角色、武器、行為等內容的擴充。此階段強調架構整合與系統設計，驗證學習者能將各模組技術融會貫通，打造出可供外掛模組擴充的遊戲。

各階段微型項目一覽

階段	微型項目	主題涵蓋重點
初階	**項目1：XML 資料驅動與 Def 定義** (含 PatchOperation)	- 資料驅動設計概念與 Def 系統架構[1] - 使用XML描述遊戲內容範例；解析並載入Def資料[6] - PatchOperation的XML節點修改機制，避免直接覆寫衝突[3]
初階	**項目2：ModContentPack 模組資源與程式碼分離**	- RimWorld Mods資料夾結構，獨立模組子資料夾設計[7] - 在Unity專案中模擬載入多個模組資料夾（資源與定義分開管理）[2] - 確認各模組資源/定義彼此隔離，互不干擾
進階	**項目3：DLL 自動載入與 Mod 類別註冊**	- 探索RimWorld自動載入DLL機制：將編譯後DLL放入模組資料夾即載入[8] - 利用反射實作模組程式碼載入：掃描模組Assemblies載入類別 - 設計 Mod 主類別（繼承自基類）註冊流程，初始化模組內容[9][10]
進階	**項目4：Harmony 前綴/後綴/Transpiler Patch**	- 理解 Harmony 庫的用途與三種Patch形式：Prefix、Postfix、Transpiler[4][5] - 撰寫前綴/後綴方法攔截並修改遊戲核心函式的行為（如改寫返回值或執行順序）[11][12] - 嘗試編寫簡單Transpiler修改IL指令（進階挑戰項目）
進階	**項目5：Mod 設定檔與 Scribe 序列化**	- 建立模組設定類別，透過 `ModSettings` 保存使用者設定[9] - 使用 RimWorld 的 Scribe 系統（Scribe_Values、Scribe_Collections 等）讀寫設定值[13] - 提供遊戲內介面讓玩家調整模組設定，並確保設定可正確保存/讀取

階段	微型項目	主題涵蓋重點
進階	**項目6：模組相依性與載入順序處理**	- 在About資訊中定義模組依賴關係，例如必需/相容的其他模組 [14] - 實作載入順序排序：依據相依性決定模組加載先後，必要時發出警告 [15] - 測試多模組情境下的衝突解決策略（如載入順序調整或停用衝突模組）
整合	**綜合專案：可擴充的極簡遊戲開發**	- 開發一款簡化版 Vampire Survivors 遊戲作為基礎 - 加入模組機制支援動態擴充角色、武器、能力等內容 - 實作至少一個外部模組（含定義檔與程式碼）驗證模組載入與擴充功能

（上述表格列出各階段的主要項目與涵蓋主題，以下將對每個項目提供教材結構草稿。）

微型實驗項目教材設計

項目1：XML 資料驅動與 Def 定義 (含 PatchOperation)

學習摘要：了解**資料驅動 (Data-Driven) 設計**在遊戲模組中的作用 – 透過**輕量數據結構**描述複雜遊戲內容，而非將邏輯寫死在程式碼中 [1] 。熟悉 RimWorld 的 **Def 定義系統**，學習使用XML檔案定義遊戲物件屬性，以及認識在沒有碰觸原始程式碼的情況下修改這些定義的技巧（**PatchOperation**）。本項目讓學生動手以XML定義一個簡單遊戲元素，並**解析載入**到Unity中使用；接著編寫一個XML補丁修改該元素的某項屬性，藉此體驗資料驅動的彈性與模組衝突的解決方案 [3] 。

項目目標：

- 能夠設計出適合特定遊戲內容的XML **定義模板**，例如定義一個遊戲角色或道具所需的屬性欄位。 [16] [17] 學會以文字格式描述遊戲物件資料結構。
- 使用C#程式碼讀取該XML檔案，解析內容並動態建立對應的遊戲物件（例如依據XML生成一個Unity中的怪物Prefab實例），驗證**資料驅動**的基本流程 [6] 。
- 撰寫一個Patch XML檔，利用 `<Patch>` 根節點和各種 `<Operation>` 子節點，對原始XML Def內容進行修改（如調整某數值或新增子節點），並透過程式加載該補丁以**更新已有的定義** [3] 。理解PatchOperation透過XPath精準選取XML節點並執行增刪改的原理。

項目說明：此項目分為兩部分：首先，學生設計一份簡單的XML定義檔（例如 `EnemyDef.xml` ），內含自訂遊戲元素的各種屬性欄位（如名稱、生命值、速度、攻擊力等）。透過**資料驅動**手法，這些屬性將由外部檔案提供，而不是硬編碼在Unity程式內 [1] 。學生需撰寫程式在啟動時讀取該XML，使用適當的解析器（如內建 `System.Xml` 或LINQ to XML）將資料轉換為程式中的物件結構，進而在遊戲中創建出對應的物件實體。例如，載入一個 `<EnemyDef>` 並生成遊戲中的敵人角色，確認其屬性符合XML設定。 [6]

接著，引入 RimWorld **PatchOperation** 的概念，準備第二個XML檔作為補丁。例如創建 `EnemyDef_Patch.xml` ，使用 `<Patch>` 根標籤，內含一系列 `<Operation>` 子標籤定義修改操作。學生將在此檔中運用**XPath**語法鎖定先前定義的節點位置，然後執行如 `<PatchOperationReplace>` 或 `<PatchOperationAdd>` 等操作，以改變原始Def的一部分內容 [3] 。舉例來說，可以使用XPath選取先前定義的敵人生命值節點，將其值從原本的100修改為150，或是新增一個新的屬性節點。程式需在載入原始Def後，再載入並套用這些Patch操作 (模擬RimWorld載入所有模組後套用Patch的流程)，最終驗證遊戲內該敵人屬性的確被更新。透過這種方式，學生將體會到PatchOperation帶來的**模組相容性**：不同模組可各自提供Patch來修改同一Def的不同部分，避免了直接覆寫整個定義時「後載入者覆蓋前者」的衝突問題 [3] 。

注意事項與常見錯誤：

- **XML 格式與編碼：**XML檔案需使用正確的格式和編碼宣告（UTF-8），並確保標籤正確閉合。初學者常見錯誤包括標籤拼寫不一致、遺漏閉合標籤或層級縮排錯亂，這些都會導致解析失敗 [18]。建議在編輯XML時使用現成的XML編輯工具或IDE確保格式正確。
- **路徑與檔案存取：**在Unity中讀取外部檔時，要注意檔案路徑的取得。可將定義檔放在`StreamingAssets`資料夾以確保輸出後仍可直接存取 [2]。常見問題是路徑拼寫錯誤或未將檔案設為正確的資源類型，導致讀取不到檔案。
- **XPath 選擇器：**撰寫PatchOperation時，XPath語法需精確匹配目標節點。如`Defs/EnemyDef[defName="Zombie"]/health`這類路徑 [19]。初學者可能在大小寫、節點名稱上出錯，或誤用`@Name`等語法導致無法匹配正確節點。建議熟悉基本的XPath選擇器用法，並充分利用RimWorld官方提供的範例作參考。
- **PatchOperation 應用順序：**若一次套用多個PatchOperation，要留意它們執行的順序對結果的影響。例如先插入節點再修改值，或條件判斷（PatchOperationConditional）的使用。如果Patch的順序不當，可能導致預期外的結果或Patch失敗。可以透過在XML中使用`PatchOperationSequence`明確定義執行順序，或將相關操作寫在同一序列中。
- **除錯方法：**當載入XML或應用Patch沒有達到預期效果時，學習使用Log輸出或除錯工具。例如輸出解析後物件的屬性值，確認是否正確讀取；或在套用Patch後重新輸出XML以檢查節點是否被正確修改。RimWorld中常用的方法是查看遊戲啟動日誌中有無Patch失敗的訊息，開發自己的項目時也可類似處理。

驗收檢查項目 (Checklist)：

- [ ] **資料定義載入：**啟動遊戲時成功讀取自訂的XML定義檔，並根據內容建立相應的遊戲物件。例如在測試場景中生成一個怪物或物品，其名稱、屬性數值與XML設定相符。
- [ ] **Patch 應用：**套用補丁XML後，原先載入的定義內容被正確修改。驗證方式：在遊戲中檢查先前創建的物件，其某項屬性是否變為補丁指定的新值（如生命值從100變為150）。若補丁為添加新元素，確認遊戲物件增加了對應的新功能或數據。
- [ ] **模組衝突測試：**模擬兩個不同補丁修改同一Def的不同屬性，順利載入且兩者修改都各自生效（前提是修改不同節點）。反之，測試兩個補丁嘗試修改同一節點的情況，確認後載入的補丁優先應用或產生衝突警告。瞭解RimWorld樣式「最後載入者勝出」或錯誤提示的情形。
- [ ] **擴充性：**快速嘗試新增另一個XML定義檔以添加**全新**的遊戲物件（如另一種怪物），無需改動程式碼即可載入生效，確保架構具備擴充新內容的彈性。同時驗證不影響原有內容。
- [ ] **代碼可讀性：**代碼結構清晰，資料載入與Patch套用的流程有良好封裝。例如將XML解析和遊戲物件生成邏輯獨立成模組管理類，日後方便重複利用於更複雜專案。並撰寫適度註解解釋關鍵步驟（如XPath字串的用意）。

推薦的線上學習資源：

- **RimWorld 官方維基：XML Def 與 PatchOperation 教程** — RimWorld維基提供了詳盡的XML定義指南與PatchOperation範例 [20] [21]。尤其參考“**Modding Tutorials/PatchOperations**”頁面，可學習各種PatchOperation類型的用途及XPath語法入門。
- **教學影片：《RimWorld Modding Guide - Basic PatchOperations》** – YouTube 上的模組教學影片，示範如何使用PatchOperation調整遊戲XML定義。透過實際案例講解XPath選取和節點替換操作，有助加深對Patch機制的理解。
- **部落格文章：AngusChan 的「如何讓遊戲支援模組開發」筆記** – 對資料驅動設計有深入淺出的說明 [22]。文章中以範例解釋如何提供外部檔案讓玩家擴充內容，以及簡單的資料格式設計與解析流程 [23] [6]。適合作為本項目的理論補充。
- **XPath 語法速查：**Mozilla 開發者網 **XPath 教學** [24]（或W3Schools的XPath指南）可協助快速掌握XPath查詢語法，避免PatchOperation中選錯節點。對初學者非常實用。

項目2：ModContentPack 模組資源與程式碼分離

學習摘要：本項目著重於**模組封裝概念**，學習如何將每個模組的資源檔與代碼**相互隔離**存放，模擬 RimWorld 對模組內容的管理方式。學生將了解一款遊戲**支援模組**的前提，是允許使用者在特定資料夾放置新增內容[2]。透過搭建類似 RimWorld `Mods/` 目錄結構的實驗，掌握**ModContentPack**（模組內容包）的設計理念：每個模組有獨立目錄，內含定義檔、素材、以及後續的程式碼檔案，遊戲在載入時能發現並讀取這些目錄下的內容[7]。此項目將延續先前的資料驅動概念，但進一步要求**同時載入多個模組**的資料，並確保彼此內容不衝突。學生也將體會到為模組建立標準結構和封裝邏輯，對日後自動化載入（例如DLL載入）打下基礎。

項目目標：

- 在專案的StreamingAssets（或自定模組資料夾）中建立**多個模組子資料夾**，每個資料夾扮演一個獨立的Mod。熟悉標準的模組目錄結構，例如包含 `About/`（模組說明）、`Defs/`（XML定義）、`Textures/`（素材）等子資料夾[25]。能說明為何需要這樣的結構（方便遊戲辨識與隔離各模組內容）。
- 編寫程式邏輯 **掃描Mods目錄** 下有哪些子資料夾（模組），並對每個模組分別讀取其資源。例如載入每個模組的Defs資料夾中所有XML，彙總到遊戲的Def資料庫中[26]。確保不同模組的定義即使類型相同也**存在各自命名空間**或ID區分，不會互相覆蓋（除非有刻意的Patch操作）。
- 演示**資源與代碼分離**的思想：此階段著重載入資源（定義檔、圖片、音效等），而程式碼部分將在下一項目處理。學生需驗證，即使在不載入模組DLL的情況下，光靠數據定義也能讓模組增添新內容（例如新增一種物品的Def及其圖像檔，遊戲讀入後即可在清單中出現新物品）。這培養對**資料驅動**與**程式碼驅動**邊界的理解。

項目說明：首先，依照RimWorld的習慣建立模組目錄結構：在遊戲的Mods主資料夾下創建若干子資料夾，每個取一個模組名稱（例如 `ExampleMod1/`、`ExampleMod2/`）。在每個模組資料夾內，添加 `About/` 子資料夾並放置簡單的 `About.xml` 檔案，填寫模組名稱、作者等基本資訊（此處可先簡化，只需能識別模組即可）。接著為模組添加 `Defs/` 資料夾，將**項目1**中製作的XML定義檔移入此處（或撰寫新的定義，如不同模組定義不同的新物件）。也可各自準備對應的素材檔，例如Textures內放與Def對應的圖像。整體結構例如[25]：

```
Mods/
 ├ ExampleMod1/
 │    ├ About/ (含 About.xml)
 │    ├ Defs/ (含若干 .xml 定義檔)
 │    └ Textures/ (含該模組資源，如圖像)
 └ ExampleMod2/
      ├ About/
      ├ Defs/
      └ Textures/
```

接下來，程式部分需要撰寫一個模組管理器 (如 `ModLoader` 類別)。當遊戲啟動時，ModLoader會搜尋 `Mods/` 資料夾下的所有子目錄，將每個目錄視為一個模組並嘗試載入：首先讀取其About.xml獲取模組名稱（可以日後用於顯示或相依性排序），然後讀取其Defs資料夾下的所有XML檔案。可以使用與項目1相同的XML解析方法逐檔讀入，並將內容追加到遊戲的Def列表中。[26] 值得注意的是，如果多個模組都定義了不同的Def（例如不同ID），則可全部載入；若發現**重名**Def，則需決定如何處理（暫定以載入順序覆蓋，或發出警告）。這裡為簡化，可規定不同模組不要定義相同ID的對象。載入完成後，讓遊戲將這些Def對象應用於場景，例如生成清單或物件，確認每個模組的內容都成功加入遊戲。學生會發現，當模組數量增加時，只需遵守目錄規範把新資源放入，遊戲就能自動識別載入，這就是**ModContentPack**架構帶來的擴充性和便利性[7]。

透過此實驗，學生實際構建了一個**簡易的模組管理系統**：具備發現模組、讀取資源、整合內容的功能。而**資源與程式碼分離**體現在：目前為止載入的都是靜態資料（Defs/XML、素材），尚無執行任意外來程式碼的風險；每個模組的資料被限定在自己的命名空間與目錄，降低了互相影響的可能。這為下一步載入DLL奠定了良好的基礎，因為我們已確定如何定位每個模組並抽象出ModContentPack的概念：未來只要擴充它來包含DLL資訊即可。

注意事項與常見錯誤：

- **目錄命名與路徑**：確保模組資料夾的名稱不含特殊字元，路徑解析時大小寫一致。在Windows系統路徑不區分大小寫，但在Unity的Android等環境可能區分，所以養成統一格式的習慣。新手常犯錯誤是模組資料夾放錯位置（非預期的Mods資料夾）導致遊戲掃描不到 [27]。遵循RimWorld的做法，將本地開發的模組放在正確的Mods目錄，而非Steam Workshop下載的模組夾，以免混淆。
- **About.xml 解析**：在本階段，可簡化解析內容，但務必至少讀取`packageId`或`name`等標識模組的欄位 [28]。很多模組排序或相依性判斷要用到`packageId` [29]。初學者有時會遺漏此步驟，使得無法識別不同模組或後續無法正確處理相依關係。
- **資源檔案類型**：若模組包含圖片等資源，注意Unity載入方式。可採用Resources資料夾或地址可讀檔案。若使用StreamingAssets路徑，可能需要用WWW或UnityWebRequest等API載入圖片轉Texture。常見錯誤是路徑拼錯或忘了轉換byte為Texture2D導致圖片不顯示 [30]。建議先專注XML等文字資源，圖片音效部分如遇困難可暫以佔位符代替，再逐步研究Unity的資源載入。
- **重覆定義與命名衝突**：當多個模組定義名稱相同的元素（例如都定義`ThingDef`名為"Gun"），在沒有Patch區分的情況下會衝突。RimWorld透過**辨識符**區分各模組Def（packageId前綴等）。在本項目，可採取約定：如`Mod1_Gun`、`Mod2_Gun`這樣在XML裡加前綴避免衝突。提醒學生注意**命名空間**的重要性，這也是為何RimWorld強調packageId [29]。
- **載入順序**：此時尚未實作依賴排序，但仍要注意載入順序可能影響結果。例如如果Mod2需要改動Mod1的內容（沒使用Patch僅靠覆寫），則Mod1的Defs須先載，再載Mod2才能覆蓋。目前階段建議學生手動控制載入順序（如按照資料夾名稱排序），並思考這種做法的局限，為後續學習相依性機制埋下伏筆。
- **性能考量**：隨著模組數量增加，初始化載入時間會變長。鼓勵學生觀察在載入多份XML時的耗時，討論優化可能（如快取已載入的模組列表，或在Loading畫面顯示進度）。儘管小型專案暫無性能壓力，但建立性能意識有助未來開發。

驗收檢查項目 (Checklist)：

- [ ] **多模組檔案發現：**遊戲啟動時正確掃描`Mods/`資料夾，列出所有模組子資料夾名稱。新增或移除模組資料夾後重新執行，確認掃描結果隨之變化。
- [ ] **模組Def載入：**每個模組`Defs/`內的XML檔皆被讀取並解析。驗證方式：將不同模組定義的遊戲物件在遊戲中實例化（例如生成不同來源模組的新物件各一個），或列印載入的Def清單，確定各模組內容均在遊戲註冊。
- [ ] **隔離性：**不同模組定義的內容互不干擾。例如Mod1新增武器“長劍”，Mod2新增武器“手槍”，最終遊戲中應同時出現兩種武器。若兩個模組定義重名項目，應有相應衝突處理策略（本階段簡單情況下，可接受後載入者覆蓋前者）。
- [ ] **易用性：**測試向Mods資料夾隨意添加一個新模組資料夾，只要結構正確（包含Defs等），無需修改遊戲主程式碼即可被載入。這說明模組擴充流程對使用者是友好的：“丟檔案進資料夾即可生效”。
- [ ] **模組資訊讀取：**成功解析每個模組的About資訊（至少名稱/ID）。可在遊戲啟動時打印模組清單，如“載入模組：ExampleMod1 (作者X), ExampleMod2 (作者Y)…”。這驗收遊戲對模組的識別能力，為後續相依性排序做準備。

推薦的線上學習資源：

- **RimWorld 維基：模組檔案夾結構指南** – 詳閱「**Mod Folder Structure**」章節 [7] [25] 。了解RimWorld對模組資料夾的標準規範（必需的About資料、各類資源子資料夾）。該指南也涵蓋了如何正確命名與組織檔案，避免遊戲無法辨識模組。
- **Unity 官方文件：Streaming Assets** – Unity Manual 關於StreamingAssets的說明 [31] 。此資料夾適合存放外部模組內容，文件中解釋了跨平台路徑以及讀取方式。對理解如何在Unity中開放遊戲目錄給玩家修改十分有幫助。
- **範例專案：Creating a Moddable Unity Game (TuriyaWare)** – 這是一篇Unity社群的博客文章，作者分享了製作可模組化遊戲的心得 [32] 。其中強調了為何需要將整個遊戲架構圍繞模組化來設計，以及資料、素材、代碼分離管理的方法 [33] 。雖然非針對RimWorld，但概念類似，可供參考如何在Unity實現模組封裝。
- **GitHub 範例：Open Source RimWorld-like 專案** – 可以查閱開源的RimWorld風格遊戲專案如FyWorld [34] [35] 。此類專案通常遵循了模組目錄劃分的理念，並在其README或Wiki中說明如何添加模組內容。閱讀他人的實作有助於鞏固對ModContentPack架構的理解。

項目3：DLL 自動載入與 Mod 類別註冊

學習摘要：進入模組技術的核心——**動態程式碼載入**。本項目教學生如何讓遊戲在執行時自動載入模組提供的外部組件（DLL檔案），從而執行玩家自訂的C#程式碼。RimWorld 的做法是：凡是放在模組 `Assemblies/` 資料夾下的已編譯DLL，遊戲啟動時都會自動偵測並載入 [8] 。我們將模擬這一流程，實作一個簡易的**模組程式集載入器**。同時，引入**Mod類別**的概念：在RimWorld，每個模組可以有一個繼承自 `Verse.Mod` 的類別，於載入時實例化以進行初始化（例如讀取設定或套用Harmony Patch） [9] [10] 。學生將學習如何透過**反射**找到模組中的主類別並創建實例，以及如何設計模組初始化的機制。這使模組不僅能帶來數據，還能執行邏輯，真正做到“程式碼即內容”的擴充。

項目目標：

- 為每個模組新增一個 `Assemblies/` 資料夾，用於存放該模組的DLL檔案。學生需準備至少一個簡單的C#類庫專案，輸出DLL放入模組中，內容可以是打印日誌或改變遊戲行為的簡短代碼。熟悉如何從Unity主程式外部引用並執行這些代碼。 [8]
- 撰寫模組載入器的程式碼，使用反射API（如 `Assembly.LoadFile` 或 `Assembly.LoadFrom`）對掃描到的每個模組DLL動態載入。確認載入成功可透過列舉Assembly中類型的方式驗證。常見做法是將DLL中包含一個已知繼承或介面的類別作為入口，以便載入後定位它。
- 設計**Mod主類別**體系：在模組DLL中撰寫一個類（例如 `MyMod : IMod` 或自定義的基底類）表示模組入口。載入DLL後，遊戲應尋找實現特定介面或具有特定命名的類（如RimWorld會尋找繼承 `Verse.Mod` 的類） [10] 。透過反射創建該類的實例，並呼叫其初始化方法（如建構子或自定Init函式）。項目目標是讓每個模組的主類別在遊戲啟動時自動執行，從而**註冊模組行為**（例如按鈕事件、Harmony補丁應用等）。這相當於模組自身的 `Main()` 函式。

項目說明：延續前一項目的模組結構，現在每個模組資料夾下添加 `Assemblies/` 子資料夾。我們可以預先準備幾個示範模組程式碼：例如一個模組的DLL內包含一個類別 `ExampleMod`，其建構子會打印「ExampleMod Loaded!」字樣；另一個模組的DLL可能有 `AnotherMod` 類，在建構時改變某個遊戲全域變數的值。把這些DLL分別放入不同模組的Assemblies資料夾。接著修改遊戲的模組載入流程：在**項目2**載入完所有Def之後，增加步驟掃描各模組 `Assemblies` 資料夾內的所有 `.dll` 檔案 [8] 。對每個DLL檔，使用C#反射動態載入Assembly（注意正確處理路徑）。載入Assembly後，可透過 `assembly.GetTypes()` 獲取其中定義的類型列表。我們需在這些類型中尋找符合**模組主類**條件的類別。例如約定所有模組主類實作一個接口 `IMod`，或直接在模組DLL內約定名稱如 `ModCore`。找到後，用 `Activator.CreateInstance()` 實例化它。此時可以將先前解析的ModContentPack資訊（如模組名稱）傳給建構子，如果約定介面可提供 `Initialize(ModContentPack pack)` 方法則呼叫之。 [10] 展示了RimWorld中 `ExampleMod` 建構子簽

名：`public ExampleMod(ModContentPack content) : base(content) ...`，我們可參考實現讓模組類別知曉自己的內容包環境。

一旦模組主類成功創建，我們就達成了**模組程式碼自動執行**。例如，若ExampleMod建構子內有`Log.Message("ExampleMod Loaded!")`，我們會在Unity日誌中看見這行訊息，證明DLL載入和實例執行成功。更進一步，模組主類可以執行任何初始化邏輯：如將事件handler註冊到遊戲事件、增加自訂介面UI，甚至應用Harmony補丁（這部分會在下一項目詳細介紹）。這個架構使得**遊戲核心與模組邏輯解耦**：遊戲只負責發現並執行模組入口，不需要提前知道模組具體做什麼。

值得注意的是安全與版本相容：本項目示範環境下，我們載入的是自己開發的範例模組DLL，相對可信。但若開放給用戶隨意放入DLL，可能遇到不相容的.NET版本或惡意代碼。RimWorld解決版本問題的方法是統一使用與遊戲相同的目標框架編譯模組，惡意代碼則較難防範（需信任模組社群品質）。此處可提醒學生意識到**模組代碼執行風險**，但不深究。最後，透過此實作，學生將掌握如何讓Unity遊戲也具備**插件機制**：任意的外部DLL只要符合約定，即可插入遊戲運行時被調用。

注意事項與常見錯誤：

- **DLL 編譯設定：**確保模組DLL的目標Framework與Unity相容（通常Unity 2021之前使用.NET Framework子集或.NET Standard 2.0）。若編譯設定不當，可能出現載入錯誤或型別不符。學生常見問題是在Visual Studio中忘記將類庫目標設為對應Unity的設定，導致`BadImageFormatException`或執行期找不到方法。解決方法：參考RimWorld模組開發建議，使用與遊戲相同的目標版本編譯DLL [36]。
- **路徑與檔案權限：**使用Assembly.Load時要給予正確的檔案路徑。相對路徑、絕對路徑處理需謹慎。在Windows桌面環境一般沒問題，但若部署到Android/iOS裝置，直讀DLL可能受到限制（移動平台的模組系統較特殊，可不在此討論）。另外，注意避免重覆載入同一DLL兩次，可在載入前檢查是否已載過。
- **類型搜尋約定：**如何識別哪個類是模組主類很關鍵。常見約定包括：1) 類名固定（如模組DLL內有與模組同名的類）；2) 實作特定接口（例如IMod接口包含Init方法）；3) 繼承某抽象基類。我們的實驗需要選定一種並文檔化給模組開發者。初學者可能會遺漏這部分約定，導致載入DLL後不知道執行哪個類。建議在教材中明確指定，例如「模組DLL須含有類別`ModEntry`實作 IMod接口」。
- **反射實例化錯誤處理：**在嘗試`CreateInstance`時，可能遇到沒有默認建構子、建構子參數不符等問題。RimWorld透過讓Mod類繼承`Verse.Mod`基類並要求實現某特定構造函數解決 [10]。我們可簡化為默認建構即可。若學生選擇需要傳參的建構子，需使用`Activator.CreateInstance(assemblyName, typeName, args)`重載並傳入參數。務必對可能的例外（Exception）加以捕獲，避免某個模組DLL出錯中斷整個載入流程。應該記錄錯誤並繼續嘗試載入其他模組，以提升健壯性。
- **重覆載入與卸載：**Unity預設不支持卸載Assembly（除了域重新載入）。因此一旦載入DLL，除非重啟遊戲，否則無法卸載或重新載入更新版本。在開發調試模組時，這意味著每次更改DLL後須重啟遊戲才能生效。提醒學生這點以免誤以為即時熱重載DLL是可行的。在RimWorld中，每次進入主選單啟動Mods配置，其實就是重啟了遊戲模組部分。
- **安全性：**載入未知DLL具有風險，本項目重點不在安全沙盒，但可附帶討論。現實中可利用AppDomain隔離或預掃描DLL限制權限等，但Unity環境下有限制。我們側重學習功能實現，安全問題點到為止即可。

驗收檢查項目 (Checklist)：

- [ ] **DLL 掃描載入：**遊戲在初始化時成功找到所有模組資料夾下的DLL檔案，並對每個DLL呼叫Assembly.Load而無錯誤。可在日誌中列出已載入的Assembly名稱，驗證包含預期的模組DLL。
- [ ] **模組主類實例化：**對於每個載入的DLL，正確找到預定的模組入口類型並創建實例。驗證方式：如果模組類建構子包含日誌輸出或修改全域狀態，檢查這些效果是否發生。如看到「ExampleMod Loaded!」等訊息，或遊戲某配置被改寫，表示模組代碼確實執行。

- [ ] **多模組執行順序：**當有多個模組DLL時，驗證它們的初始化順序是否按照掃描順序進行（或依相依性，若已實作排序）。例如ModA和ModB都有輸出訊息，確認日誌中順序與預期一致。如果需要特定順序，可在後續項目透過相依性解決，此處接受預設載入順序。
- [ ] **模組行為作用：**模組DLL內可以執行對遊戲的實質擴充。例如一個模組在初始化時登錄一個新的按鍵指令或菜單項，操作對應功能。測試該功能是否隨模組載入而出現、移除DLL後消失。這說明模組程式碼成功掛接到遊戲系統中。
- [ ] **錯誤處理：**模擬一個損壞或不符合約定的DLL檔案（例如不包含模組主類）。載入器不應因例外而崩潰，應捕捉錯誤並報告。例如日誌警告「Mod XYZ: 找不到入口類，略過載入代碼」。確保不影響其他模組載入。

推薦的線上學習資源：

- **RimWorld 維基：Setting up a Solution** – RimWorld模組編程入門教程，說明如何建立模組的C#解決方案 [37]。其中涉及將RimWorld的Assembly作引用、匹配遊戲版本編譯等細節，這對了解DLL載入相容性很有幫助。
- **Harmony 官方文件：**儘管Harmony詳細內容在下一項目才用到，但其**靜態構造註冊**方法值得參考 [38]。RimWorld許多模組在載入時利用`StaticConstructorOnStartup`屬性自動執行代碼。閱讀Harmony作者的文檔有助於理解模組代碼何時執行。
- **C# 反射與動態載入指南：**Microsft Docs 上關於反射的章節，特別是`Assembly.LoadFrom`與`Type.GetType`用法。或者參考博客《在運行時載入C#插件》之類的文章，獲取在Unity中使用反射的經驗談。
- **GitHub 範例：HugsLib** – HugsLib是RimWorld社群提供的模組支援庫，其代碼開源。可瀏覽其ModuleLoader實現方式，觀摩成熟模組載入框架怎樣管理DLL、調用靜態初始化等。HugsLib還提供了自己的Mod基類，可作為我們項目Mod類設計的靈感。

項目4：Harmony 前綴/後綴/Transpiler Patch

學習摘要：本項目聚焦於**Harmony**庫的使用，它是目前修改RimWorld等Unity遊戲**核心邏輯**的最佳實踐方式 [39] [40]。學生將學習Harmony的三種主要Patch方法：**Prefix**(前綴)、**Postfix**(後綴)和**Transpiler**，理解它們的作用範圍與適用情境 [4] [5]。透過實作與測試簡單的Patch，體驗如何在不更改原始碼的情況下攔截或調整遊戲函式的行為。例如，在原方法執行前插入代碼、在之後追加代碼，甚至直接修改函式的IL指令。學生將在小型測試場景中運用Harmony修改自己遊戲的某個方法（或甚至RimWorld的代碼，若已安裝），並學習驗證Patch效果的技巧。此項目培養學生對**模組衝突**和**相容性**的更深理解：Harmony補丁通常比直接改原始函式更具相容性，能與其他模組並行存在 [41]。

項目目標：

- 熟悉Harmony基本用法：會安裝Harmony庫（0Harmony.dll）並在模組專案中引用。瞭解**不應重覆包含Harmony DLL**於每個模組中，而是由遊戲或一處集中管理，避免版本衝突 [42]。本項目可直接將Harmony作為遊戲執行時依賴。
- 能夠撰寫**Prefix Patch**：選擇遊戲中的一個目標方法（例如角色受到傷害的函式），編寫一個前綴方法，以Harmony註解或手動調用方式將其與目標方法綁定 [11]。理解prefix可選擇回傳bool決定是否跳過原方法執行 [4]。測試prefix的效果，例如條件符合時跳過原邏輯達成某種改變。
- 能夠撰寫**Postfix Patch**：對另一個目標方法編寫後綴，在原方法執行完畢後取得其結果（可能透過ref獲取__result）並加以修改 [43] [44]。驗證後綴確實改變了最終結果或執行了附加行為，而且不影響其他模組的後綴執行順序（Harmony保證所有Postfix都會執行） [45]。
- 初步了解**Transpiler**：雖然Transpiler較複雜，本項目鼓勵學生**嘗試**寫一個簡單Transpiler，例如將目標方法的某個常數乘2（透過修改IL碼實現）。重點在於體驗流程而非精通，可使用Harmony提供的`CodeInstruction`列表進行模式匹配替換。若困難，可選擇性略過Transpiler細節，但需理解其用途是修改方法內部邏輯，是功能最強也最危險的patch方式 [46]。

- 建立**Patch 測試**習慣：透過日誌或Unity的UI反饋，證明補丁是否生效。例如在Prefix中打印log確認被調用次數，在Postfix中輸出原始結果和修改後結果。學會使用Harmony的除錯功能（如`HarmonyInstance.DEBUG = true`查看補丁日誌）[47]來診斷patch未奏效的原因（可能方法名錯誤或簽名不符）。

項目說明：在開始動手前，先讓學生理解**為何需要Harmony**。舉例：如果我們想讓遊戲中的角色移動速度加倍，傳統做法可能需修改角色更新函式的程式碼。但若無法改動核心程式，模組能做的是**插入一段代碼**在速度計算處將結果x2。Harmony使這成為可能：它允許我們對目標方法附掛一個前置或後置函式。在前置中可改變原方法的參數甚至選擇不執行原方法，在後置中可拿到原方法的返回值並調整[4][5]。

實作步驟：將Harmony　DLL加入專案（通常將0Harmony.dll放入遊戲主程式的Plugins或模組中依賴，但**切記**只用一份Harmony[42]）。在前面項目3的模組DLL專案中，引用Harmony。然後撰寫一個靜態類含有Patch方法。例如：

```
[HarmonyPatch(typeof(PlayerController))]
[HarmonyPatch("GetMoveSpeed")]
class Patch_PlayerSpeed
{
  static void Postfix(ref float __result)
  {
    __result *= 2f;
  }
}
```

這段Pseudo-code示範了一個後綴，目標是`PlayerController.GetMoveSpeed`方法，將其返回值加倍。在模組主類的初始化（或靜態構造）中，執行Harmony啟動程式碼：

```
var harmony = new Harmony("com.mygame.mod.example");
harmony.PatchAll();
```

這會自動應用當前Assembly中定義的所有[HarmonyPatch]類別。若不使用屬性，也可以手動使用`harmony.Patch(originalMethod, prefix, postfix, transpiler)`來指定。應用後，運行遊戲，應觀察角色速度確實變快了兩倍，驗證Postfix成功影響結果。再例如寫一個Prefix：

```
[HarmonyPatch(typeof(Enemy))]
[HarmonyPatch("TakeDamage")]
class Patch_EnemyDamage
{
  static bool Prefix(Enemy __instance, ref int amount)
  {
    // 讓敵人名為Boss的不受傷害
    if(__instance.name == "Boss")
    {
      amount = 0;
      return true; // 執行原方法但傷害已改為0
    }
    return true; // 繼續執行原本方法
```

```
    }
}
```

這前綴會在Enemy物件執行TakeDamage前被調用，可以修改傳入參數`amount`。如上例，若敵人是Boss，我們把傷害設為0（等於無敵）。`Prefix`返回true表示仍繼續執行原方法（這裡選擇讓原方法跑，但傷害值已改），若返回false則原方法完全跳過[4]。測試時，攻擊Boss檢查其生命值是否維持不減以驗證補丁。

Transpiler相對複雜，可以挑戰有興趣的學生：例如選一個簡單方法，編寫Transpiler將其中IL碼的加法指令換成減法等。Harmony作者提供了一些筆記[48]和社群有教程可參考。我們可引導學生至少讀懂Transpiler的概念：它其實就是對方法的MSIL指令列表做Find/Replace操作。

最後，在測試過程中強調**相容性**：如果另一個模組也Patch了同一方法會怎樣？通常多個前綴都有機會執行（除非某前綴返回false提前跳過原方法，那之後的前綴就沒機會了，要小心這種設計）[43]。而後綴基本都會執行，只是執行順序由Harmony決定（預設先Patch先執行）。Transpiler多個的情況則比較複雜，需要仔細協調。我們可在教材中簡述，如RimWorld社群建議**盡量使用Postfix實現效果**以保持最大相容性[49]。Prefix僅用於必要時（且避免`return false`除非確定安全）[4]。Transpiler則不到萬不得已不用。這樣的Best Practice也讓學生明白，在實際開發模組時，選擇適當的patch類型能減少與其它模組衝突。

注意事項與常見錯誤：

- **方法簽名匹配：**撰寫Prefix/Postfix函式時，其參數類型和數量需與目標方法吻合，特別是實例方法的`__instance`類型、靜態方法沒有實例、以及`__result`的類型要正確（Postfix用）。學生初次常犯的錯是簽名不對導致Patch無效。如目標方法有參數(int x)，Prefix若漏寫`ref int x`就對不上。參考Harmony維基或使用工具如HarmonyX Analyze可以驗證簽名是否正確。
- **多次Patch相同方法：**重覆執行PatchAll可能多次Patch同一方法，引發錯誤。保證每個方法只Patch一次。同時，也提醒不要在不同模組裡Patch彼此的Patch類，這樣過於複雜且難除錯。
- **調試補丁：**使用Harmony提供的日誌和容錯機制。將`Harmony.DEBUG`設為true時，Harmony會輸出詳細的補丁資訊檔案，包含哪些方法被Patch以及成功與否[47]。如果patch沒生效，優先檢查這些日誌。還可以在遊戲中使用反射檢查Method是否被打上Harmony標記（通常無必要）。
- **避免過度Patch：**在教材中可重申**不要濫用Harmony**[50]。能透過更高層手段（如修改Def數據或用遊戲提供的掛鉤）實現的，不一定要用Harmony。特別是Transpiler代碼難以維護，也最易隨版本更新失效[46]。這一點讓學生明白，Harmony是強力但有成本的工具，用前要權衡。
- **Harmony版本：**注意Harmony有1.x和2.x版本，RimWorld 1.1之後使用Harmony 2.x。確保引用正確版本，模組之間應一致。避免自己捆綁舊版Harmony到模組導致和遊戲內置版本衝突[42]。本實驗環境中，由我們控制，只載入一次最新Harmony。
- **授權與貢獻：**提醒學生Harmony是開源庫，有詳細文檔和社群支持。如果日後想深入，可閱讀Patrice (Harmony作者) 的教程和試著貢獻或插件。這也建立良好的開源觀念。

驗收檢查項目 (Checklist)：

- [ ] **Harmony 初始化：**遊戲啟動時Harmony成功載入且啟用。驗證方法：無錯誤日誌關於Harmony版本或載入衝突，並確認Harmony相關操作（如PatchAll）被調用一次。
- [ ] **Prefix 功能驗證：**實際效果符合預期。例如套用Prefix使某角色無敵，在遊戲中攻擊該角色，其生命值不減少，證明Prefix修改了參數或跳過了原執行。檢查其他角色仍正常受傷，表示Prefix條件判斷正確生效沒有影響全局。
- [ ] **Postfix 功能驗證：**觀察被Patch的方法返回值或狀態確實被Postfix更改。例如我們將玩家移動速度加倍的Postfix，進入遊戲比較有無模組時角色移動的快慢差異，或透過除錯輸出原本結果與新結果比較。

- [ ] **Transpiler (如實作則驗收)：**若學生嘗試Transpiler，在小規模測試中確認效果。例如把一段方法的+運算改-，那被Patch後的功能輸出與原本相反。如Transpiler沒成功，分析IL日誌找到問題所在。這一項對不熟IL者可選擇性跳過或僅口頭講解，不強制。
- [ ] **多補丁共存測試：**模擬兩個不同模組Patch同一函式的情況（或在同一模組內連續Patch兩次）。觀察遊戲是否仍然運作、兩個Patch是否都執行。預期結果：多Postfix全部執行，Prefix若都return true則都執行且原方法執行一次。如有衝突（例如兩個Prefix其中一個return false），需解釋這種衝突的處理（後者可能無效）。該測試有助於理解模組之間需要協調避免互相搶先破壞邏輯。
- [ ] **日誌無誤：**檢查Harmony除錯日誌或遊戲控制台，沒有patch失敗或異常錯誤。若有警告/錯誤，需分析是否由Patch簽名不符或目標方法不存在導致，並修正程式後重測。

推薦的線上學習資源：

- **Harmony 官方Wiki與檔案：**Harmony作者的網站提供詳細的文檔和指南 [51]。其中“**Basics**”和“**Patching**”章節解釋了Prefix/Postfix/Transpiler的原理 [52]。閱讀官方指南能全面了解Harmony的能力和限制，是模組開發者必讀資源。
- **RimWorld 維基：Harmony 教程** – RimWorld維基的「**Modding Tutorials/Harmony**」頁面專門講解在RimWorld環境使用Harmony的注意事項 [40] [50]。內含一些Pitfalls（陷阱）討論，如避免過度使用、如何處理返回值等實用建議。亦有Links部分列出了HarmonySteam模組頁和NuGet下載 [53]。
- **Stardew Valley 等其他遊戲的Harmony實例：**例如Stardew Modding Wiki上的Harmony補丁教學 [54]。不同遊戲但概念類似，可看看他們如何用Prefix/Postfix解決問題。這有助開闊思路，明白Harmony運用不局限於RimWorld。
- **社群範例和討論：**前往RimWorld官方Discord的#mod-development頻道，或閱讀Ludeon官方論壇中Harmony相關討論帖。很多Modder會分享補丁經驗和疑難解答（例如有人問「為何我的Patch不生效？」得到的答覆往往揭示了常見錯誤）。透過他人的QA學習，能加深對Harmony細節的理解。

項目5：Mod 設定檔與 Scribe 序列化

學習摘要：在模組功能越來越豐富後，許多情況下需要讓使用者**調整模組的參數**或做個性化配置。例如一個模組添加了新的武器，玩家也許希望開關某些功能或調整威力。為此RimWorld提供了**Mod Settings**機制，允許模組作者定義可儲存的設置值，並在遊戲的「模組設定」介面中提供UI讓玩家修改。這背後涉及**序列化**技術：將設置存入檔案、讀取回來應用。RimWorld使用自家的**Scribe**系統進行序列化，封裝了保存/加載的細節。學生在本項目將學習如何創建自定的ModSettings類別，利用RimWorld類似的方法（ExposeData或Scribe調用）將配置持久化 [13]。此外，還需設計一個簡單的**設定UI**，可以在遊戲中調整這些值。總之，本項目讓學生掌握為模組添加**可配置選項**的流程，並了解序列化的一般原理。

項目目標：

- 定義一個**模組設定類別**，例如 `MyModSettings`，包含若干公開屬性對應可調整的設置項（如 `bool enableFeatureX`, `float someMultiplier` 等) [55]。讓此類繼承一個基類（若模擬RimWorld可自行定義 `ModSettings` 基類）以統一管理。實作一個 `ExposeData()` 方法，在其中使用我們的序列化工具對每個屬性進行保存/加載。可以仿照RimWorld的用法：利用靜態方法如 `Scribe_Values.Look(ref myValue, "myValueKey", defaultValue)` 來保存或讀取值 [13]。
- 修改模組主類，使其持有一份設定類別實例。例如在模組類建構時創建 `settings = new MyModSettings()`，並嘗試從磁碟讀取先前保存的值來初始化它（或如果沒有檔案則使用預設值）。確保模組類別能訪問並使用這些設定，例如根據設定開關功能。對應到RimWorld，其Mod類的建構子常呼叫 `GetSettings<ModSettingsType>()` 以取得已載入或新建的設定 [56]。
- 設計一個**設定介面UI**。在Unity中，可以簡單地在遊戲啟動介面或者模組介面中放置幾個控制項（如Toggle、Slider），讓玩家可以改變設定類的對應屬性值。為了一致性，可在模組主類提供一個方法如 `DoSettingsWindowContents(Rect inRect)` 來繪製UI [57]。RimWorld中使用了

`Listing_Standard` 輔助繪製設置項 [58]，我們可以自行用Unity的Immediate UI或Editor GUI來實現。同學應確保當玩家操作UI時，實時更新我們的設定對象的值。

- **保存與載入**功能：當玩家調整了設定並確認/退出介面後，我們需要將新值寫入檔案保存。可在模組介面關閉時觸發保存，也可提供一個“保存設置”按鈕。保存時調用我們的ExposeData或類似序列化方法將值寫到如 `ModSettings/MyMod.xml` 文件。下次啟動時，模組主類應從該檔讀取並恢復值，做到**持久化**。測試透過多次啟動遊戲，確認設置改動能夠保持。

項目說明：首先，引入一個簡單的序列化工具。在Unity環境下，可使用JSON、XML、二進制皆可。為了體會RimWorld的風格，我們實作一個類似Scribe的靜態幫助類，例如 `ScribeManager`，提供方法如 `SaveValue<T>(string key, T value)` 和 `LoadValue<T>(string key, ref T field, T defaultValue)`。內部可以用 `PlayerPrefs` 作簡單保存，或用System.IO寫文件。由於RimWorld的存檔系統較複雜，這裡我們簡化為Key-Value保存即可（重點是概念）。接著創建 `MyModSettings` 類，加入需要儲存的欄位並寫ExposeData：對每個欄位調用我們的保存/讀取方法。RimWorld的Scribe有不同類別（Values, Collections, Defs等） [59] [60] 但我們不必實作全部，只要處理基礎類型。ExposeData在序列化系統工作時被呼叫。可以在模組主類手動呼叫（如在遊戲退出或模組取消時調用一次保存）。

在模組主類中，加一屬性 `settings` 類型為MyModSettings。啟動時嘗試讀取檔案到settings（如果沒有則用建構時預設值）。這類似RimWorld的流程：模組建構子透過 `GetSettings` 從文件載入或創建新設定 [56]。之後，將settings內的值應用於模組功能：例如如果有 `enableFeatureX = false` 則不執行某段代碼。

UI方面，可以在遊戲主選單新增一個「模組設定」按鈕列表，列出目前載入的模組，點選後展示對應模組的設定項。由於我們示範的模組不多，可以簡化處理，只做自己模組的UI。使用Unity的UI元件：比如製作一個面板，其中有對應settings屬性的輸入元件（如Toggle綁定到settings.enableFeatureX）。運行時把Toggle的狀態和settings同步。當UI關閉時執行保存。RimWorld中Mod設定UI由每個Mod類別的 `DoSettingsWindowContents` 負責繪製 [57]，遊戲框架會呼叫這個方法。我們在實驗中可以自己管理UI，所以也可不仿這一結構，而是直接在Update監聽UI事件。

將設置保存到文件時，可考慮使用XML格式以統一風格。例如存成：

```
<ModSettings>
  <enableFeatureX>true</enableFeatureX>
  <someMultiplier>1.5</someMultiplier>
</ModSettings>
```

如此下次讀取解析XML節點賦值即可。這部分實現可以讓學生自由發揮，用他熟悉的方法存檔也行。

整個流程測試：修改設置->保存->重啟遊戲->設置仍然保持->功能按新設置運行。舉例：原先enableFeatureX默認false，玩家開啟後保存。重進遊戲觀察某功能（FeatureX）是否啟動。如果是，就成功了。

透過此項目，學生將掌握**遊戲模組如何提供可配置選項**。這也是專業遊戲開發的重要內容，因為讓玩家自定義體驗可以顯著增加模組的友好度與實用性。此外，序列化的知識對日後處理遊戲存檔、資料持久化也很有用。

注意事項與常見錯誤：

- **序列化格式選擇：**RimWorld用了自訂的Scribe做保存。我們可以選擇簡單路線用JSON或XML庫。學生可能會因不熟悉檔案IO導致保存失敗，比如忘記給檔案路徑寫權限或在編譯後路徑改變。建議初步用Unity的 `PlayerPrefs`（其實是註冊表）驗證邏輯，再換成檔案保存增加難度。
- **ExposeData調用時機：**在RimWorld，遊戲在特定時機（打開模組設定介面、退出遊戲時）調用各Mod.Settings的ExposeData進行保存 [61]。我們若模擬，需自己決定何時保存。一種簡單做法：每次設

置變更立即保存。但頻繁IO可能不理想。或者在玩家離開設定介面時一次性保存。學生常忘記調用保存導致改了設定沒寫入檔案。需強調”改了不存=白改“的概念。

- **默認值處理：**ExposeData通常允許設置預設值以在讀取不到時使用 [62] 。要提醒學生提供合理的默認值，並在序列化方法中處理首次沒有存檔的情形。如果沒注意，可能導致第一次載入時因沒有檔而報錯。
- **UI同步：**確保UI元件的值和settings物件的值雙向同步。如果只更新UI顯示而忘了改settings，或反之，則保存時會出問題。可以在UI初始化時從settings讀值，變更時立即寫回settings。這在實作上要小心避免搞混。
- **多模組設定檔區分：**如果未來有多個模組，都存自己的設定，需要區分檔名或路徑。例如可用模組packageId命名設定檔。初學者可能把不同模組設定寫到同一檔，後果是互相覆蓋。RimWorld透過LoadedModManager管理各mod settings並以packageId區隔 [63] 。我們也應至少用不同檔名區別。
- **HugsLib 等工具庫：**RimWorld mod界有HugsLib庫簡化了設定管理，它提供了更易用的Attribute自動綁定UI。但是本項目我們從頭做，學生可能會好奇有無現成工具。可以告知有這類庫，但為了深入理解，暫時不用它們。在實務中可考慮採用以節省時間。

驗收檢查項目 (Checklist)：

- [ ] **設定屬性生效：**模組提供的設定項能影響模組行為。測試更改每一項設定對應的功能開關或參數是否隨之改變。例如有一個「啟用高畫質特效」設定，開關關聯到模組中特效的顯示。切換後在遊戲中能立即或稍後看到特效開啟/關閉的變化。
- [ ] **設定UI可用：**在遊戲中成功打開模組設定介面，UI元素對應正確的設定值（如Checkbox勾選狀態與布林值相符，Slider位置對應數值大小）。操作UI能實時更新內存中的設定對象。UI布局無重大問題（不用太美觀但要能正常交互）。
- [ ] **保存與載入：**調整設定後退出遊戲再進入，之前的改動仍然存在。可逐項驗證：將某值改成非預設，重啟後觀察UI顯示和模組行為都保留了該值，證明序列化成功。也可直接打開保存的檔案檢查內容格式正確 [64] 。
- [ ] **默認值處理：**刪除設定檔再進入遊戲，模組能以預設值運作且不會拋錯。此情況模擬首次安裝模組或重置設定。所有設定應重置為代碼中定義的默認值而非殘留舊值。
- [ ] **異常處理：**故意損壞設定檔（如修改XML使格式錯誤）後啟動，模組能檢測到並回退到預設值或提示錯誤，而不會造成整個遊戲崩潰。這驗證對序列化讀取錯誤有適當處理。

推薦的線上學習資源：

- **RimWorld 維基：Mod Settings 教程** – RimWorld官方維基“**Modding Tutorials/ModSettings**”詳細介紹了如何在模組中實現設定保存 [9] 。包括需要兩個類（Mod和ModSettings）、如何ExposeData、如何寫DoSettingsWindowContents等步驟 [57] 。對照該教程能檢查我們項目的實現是否遺漏重要部分。
- **RimWorld 存檔系統解析 – “Saving For Dummies”** [59] [60] – 這是RimWorld社群整理的存檔教學。雖然涵蓋範圍比ModSettings廣，但其中對Scribe幾種保存方法有淺顯解釋（例如Scribe_Values如何用 [65] ）。透過此資源可以深入理解RimWorld序列化框架，也能學到一些最佳實踐（如保存集合、引用的技巧 [66] ）。
- **Unity ScriptableObject / PlayerPrefs 教程：**如果對使用Unity內建手段實現配置有興趣，可參考Unity官方或部落客介紹的ScriptableObject用於保存配置的方法，以及PlayerPrefs的簡單存取。這些不是RimWorld的做法，但作為遊戲開發者通用知識值得了解，日後遇不同情境可靈活運用。
- **範例代碼：GitHub 上的模組設置實現** – 搜尋一些開源的RimWorld模組，看它們如何管理配置。例如 **LWM.DeepStorage** 模組的設置類 [67] [68] 。觀摩實際模組的Settings代碼，可以發現他們在ExposeData裡保存了哪些內容、用什麼UI控制。這能提供我們項目之外更實際的參考範例。

項目6：模組相依性與載入順序處理

學習摘要：最後一個微型項目處理模組系統中非常重要但也較複雜的部分：**模組間的相依性**和**載入順序**。當遊戲擁有眾多模組時，往往會有某些模組需要在另一模組之上運作（例如擴充另一模組的內容），或者兩個模組

修改了同一內容需要特定順序才能共存。RimWorld通過在 `About.xml` 中讓模組宣告**需要哪些其他模組**，以及**應該在誰之前/之後載入**來解決這個問題 [69] [15]。遊戲會根據這些宣告自動排序模組並在有衝突時發出警告。學生在本項目將學習如何解析模組的相依性資訊，並實作一個簡易的排序演算法決定模組載入順序，以及模組衝突的檢測與提示。這將確保在最終整合專案中，多模組環境能**穩定有序**地載入，不會因順序不當導致功能失效或錯誤。

項目目標：

- 擴充模組描述格式（About資料）以包含相依性宣告。例如在每個模組的About.xml加入 `<modDependencies>` 節點，列出該模組所需的其他模組的packageId清單 [70]。也可加入 `<loadBefore>` / `<loadAfter>` 等順序建議節點 [15] [71]。學生需要修改前面項目2的解析程式，將這些新欄位讀取出來並存入模組資訊物件。
- 根據收集到的相依性資訊，實作一個**模組排序算法**。可採用拓撲排序（Topological Sort）處理依賴關係：建構有向圖，節點是模組，邊表示依賴。然後生成一個依賴順序，確保每個模組在其所需模組之後載入 [72]。同時考慮loadBefore/loadAfter的偏好，這可轉換為附加的順序約束。在簡化情況下，如果無循環依賴，算法能給出一個可行順序。驗證排序正確的方法是：如果Mod B 依賴Mod A，排序結果中A一定在B之前。
- 處理**循環依賴**或**缺失依賴**等異常狀況。若兩模組互相依賴，或依賴一個沒有安裝的模組，應檢測並給出警告而非陷入死循環。RimWorld在缺失依賴時會在介面上標示紅字警告玩家 [69]。我們可以模擬：將此資訊輸出日誌或在UI上提示“模組X需要模組Y，但後者未載入”。對循環依賴，可簡化為打破循環並提示（例如隨意決定一個順序，但報告衝突給用戶）。
- 尊重**衝突標記**：有些模組會標記不相容的其他模組或必須在其前後的嚴格順序（RimWorld有 `<incompatibleWith>` 和 `<forceLoadBefore>` 等標籤 [73] [74]）。我們未必全部實作，但至少理解概念並在資料結構上預留可能。例如可支持一個簡單清單“不可與X同時載入”，當檢測到同時存在就提示用戶關閉其中之一。這些屬於進階挑戰目標，鼓勵有能力的學生實現。

項目說明：以項目2和3建立的模組系統為基礎，現在給每個模組定義**元數據**表示依賴。打開之前模組的About.xml，在 `<ModMetaData>` 中加入：

```
<modDependencies>
  <li>
   <packageId>AuthorName.OtherMod</packageId>
   <displayName>Other Mod</displayName>
  </li>
</modDependencies>
<loadAfter>
  <li>AuthorName.SomeLibraryMod</li>
</loadAfter>
```

類似上述，宣告當前模組需要 `OtherMod`，且應在 `SomeLibraryMod` 之後載入。如果相依的是特定版本，可加版本欄（本實驗可不深入版本號）。調整我們的ModLoader：解析About.xml時，把這些子列表讀出，存入我們的 `ModInfo` 物件，例如增加字段 `requires = []`、`loadAfter = []` 等。然後在正式載入模組內容（Defs/DLL）之前，先對所有ModInfo跑一個排序程序。可以使用**拓撲排序**：實作一個簡單DFS或Kahn's algorithm都可。按requires關係構造有向圖，找出一個滿足所有依賴的順序。

例子：有ModA, ModB, ModC。若ModC requires ModA，ModB loadAfter ModA，則圖中A->C, A->B。拓撲排序可能給出A, B, C。再考慮ModC沒有特別要求B，所以C可以在B後也行，但是因為我們沒特別規定C和B的順序，可能排序算法會把C在B前也可。這時就要看loadAfter等約束：如果ModC loadAfter ModB，我們就再加一邊B->C。總之，把requires和loadAfter兩種都當作“必須在之前”的約束處理。loadBefore則反過來可以轉

成“此mod應在列出的mod之前載”，等價於對方有loadAfter我們。最後得到的順序列表，我們按此順序執行模組的DLL載入和初始化，而非之前簡單的字母序或發現序。

接著處理**缺失依賴**：假如ModC需要ModX，但ModX不在已載列表。我們在解析時就能發現，因為遍歷ModC的requires能找不到對應packageId的ModInfo。對這種情況，可以做兩件事：1）在排序時忽略該依賴（否則圖中有個節點X無法處理）；2）記錄一個警告。最後排序完成或在開始載入前，向玩家或日誌輸出：“警告：ModC 所需的 ModX 未載入，可能導致錯誤。”。RimWorld就是在介面上標紅並且允許玩家依然嘗試運行[75]。我們可以選擇繼續載入ModC，但標記它狀態不正常。或者乾脆不載也可以，看策略取捨。教育上最好提示但讓模組繼續載，除非我們實現更複雜的禁用邏輯。

衝突（incompatible）：如果我們實現這部分，可能是在About.xml裡檢查如果ModA標明incompatibleWith ModB，而現在兩者都在列表中，那就在啟動時報錯建議關掉其中之一。這類和玩家選擇相關，我們傾向於提示即可，不自動幫關閉（RimWorld本身也只是警告還是允許載入）。

當完成排序並處理了潛在問題後，我們按照得到的順序進行模組載入（Defs -> DLL -> 初始化）。可以打印出實際順序供檢查。用多組不同依賴關係的模組測試：如ModAlpha 無依賴、ModBeta 需Alpha、ModGamma 無依賴但要求在Alpha後。正確順序應Alpha -> Beta -> Gamma（Beta需要Alpha先，而Gamma的loadAfterAlpha也會實現）。再換一組有循環的：ModP requires ModQ, ModQ requires ModP，排序算法會發現循環無解。這時我們至少要**偵測**到。可用簡單的cycle檢測（拓撲排序時剩下未排序的就是循環）。對循環我們可以任意定順序但要報警。比如就按檔名順序P,Q，但提示“P和Q互相依賴，載入順序已強制設定但可能不穩定”。讓學生理解這種情況最好是模組作者避免，作為框架只能盡量處理。

通過這項目，學生將明白**為何要有模組依賴宣告**以及**如何利用它**提高遊戲穩定性。這在規模稍大的模組生態中是不可或缺的，否則玩家無從知道正確的載入順序而可能導致故障。同時也體驗了解決依賴關係的一般算法（這與軟體包管理的依賴解析類似），擴展了對計算機科學圖論應用的視野。

注意事項與常見錯誤：

- **packageId vs 名稱：**依賴應該基於唯一ID而非僅名稱[28]。RimWorld使用`packageId`（通常格式Author.ModName）作為依賴標識[29]。學生在實作時別用模組顯示名稱比對，因為名字可以重覆且有本地化。務必使用機器可讀的ID。
- **解析順序**：要在排序前拿到所有模組的依賴資訊。因此About.xml解析不能等到排序後。幸好我們已經在載入前讀取About，所以沒問題。但要提醒不要邊排序邊動態去讀依賴，那樣可能漏掉或亂序。
- **排序算法正確性：**小心實現拓撲排序避免無限迴圈。初學者實作DFS容易忘記標記狀態導致重複遍歷。建議用Kahn算法較直觀：每次選取無前置依賴的mod加入順序並移除它的邊，迴圈直到無mod剩。若還有剩表示循環。這樣也方便偵測循環。
- **UI 提示**：在玩家角度，最好能在模組列表UI中就顯示依賴關係與錯誤。RimWorld的Mod介面在模組選中時會列出其需要和衝突項並標紅缺失依賴。實驗項目不要求做完整UI，但可以在Console日誌或簡易介面上輸出相關資訊。例如“ModA will be loaded before ModB (dependency)”。這有助於除錯和讓用戶了解順序結果。
- **版本相依**：進階討論是，不同遊戲版本或模組版本的依賴，如requires v1.2以上。這太複雜可略。但讓學生知道有這情況即可。如果有時間可說明RimWorld `<supportedVersions>`標籤[76]用於指明模組適配的遊戲版本，雖不直接影響順序但影響相容性警告。
- **自動排序 vs 手動干預**：我們演算法排序屬自動，但玩家可能想手動調整。RimWorld允許用戶拖拽排序，然後在存的ModsConfig.xml裡記錄。如果我們最終專案也計畫有模組管理介面，要考慮讓用戶覆蓋自動排序。但在本項目重點是演算法，所以此點可以在綜合專案討論中提及，不要求現在實現。

驗收檢查項目 (Checklist)：

- [ ] **正確排序**：設計幾組模組依賴測試，用不同拓撲關係驗證排序結果。每組測試比較我們演算法給出的順序與預期是否一致。特別檢查：所有requires約束均被滿足（依賴都在前），所有loadAfter約束亦滿足（指定在後的確實在後）。
- [ ] **缺失依賴警告**：移除一個被其他模組需要的模組，啟動時應看到對應警告。如“Mod X 需要 Mod Y (未安裝)”。確認載入流程沒有中斷，相關的Mod X仍嘗試載入（或根據策略跳過，但需一致）。
- [ ] **循環依賴檢測**：構造兩個或三個模組環環相扣的情況，啟動時我們的系統應能偵測到迴圈並發出通知。例如在日誌中輸出“依賴關係存在循環：ModA ↔ ModB，已自動處理順序但可能不穩定”。並驗證最終仍給出一個順序（哪怕隨機）。
- [ ] **不相容模組警示**：如果實作了incompatible處理，測試同時載入互斥的兩模組時收到提示。例如“ModC 與 ModD 不相容，建議只啟用其一”。確認我們沒有繼續載入兩者導致潛在衝突（或者即便載入也提醒用戶）。
- [ ] **排序穩定性**：增加無依賴關係的模組，確認它們相對順序在多次執行中保持一致（比如都沒有相依則可按字母順或發現順排列）。這不是硬性要求，但穩定的排序有助於除錯（玩家期望一樣的輸出順序）。我們的算法如Kahn可能有多解，最好固定一個選擇策略，例如字母序選擇下個節點，來確保穩定。

推薦的線上學習資源：

- **RimWorld 維基：About.xml 相依性欄位** – 參閱維基中About.xml格式的“Optional Tags”部分 [77] [78]。詳細列出了 `<modDependencies>` 以及 `<loadBefore>/<loadAfter>` 等用法，還有 `<forceLoadBefore/After>` 和 `<incompatibleWith>` 的作用 [73] [74]。透過官方文檔瞭解這些欄位的設計意圖，有助於我們在自己的系統中實作對應功能。
- **依賴解析算法教程：**計算機科學領域有大量資料講述拓撲排序與有向無環圖。有些中文教程或教材章節專門例子是課程排程或任務調度，可參考其步驟來實現我們的模組排序。了解拓撲排序的原理也為學生拓展了算法知識。
- **社群工具：Mod Manager (Fluffy)** – RimWorld Mod Manager模組是社群著名的模組排序工具。雖然其代碼較複雜，但文檔和介紹中可能有提到排序邏輯和衝突檢測的思路 [79]。可以閱讀Fluffy Mod Manager的使用說明，看看它如何提示用戶關於依賴/衝突，這對設計使用者介面很有啟發。
- **實際案例討論：**在Reddit或論壇上，有玩家討論模組排序的重要性和遇到問題的經驗 [80]。例如“**How important is mod order?**”等帖子強調了載入順序對遊戲穩定的影響。一些回答會解釋模組載入的基本原則和軟體衝突的根源。通過這些討論，學生可以從用戶角度理解做好依賴管理的價值，以及失敗時的後果。

總結型整合專案：可擴充的極簡 Vampire Survivors 類遊戲

專案概述：在完成上述初階和進階階段的學習後，最後的整合專案將帶領學生將所有知識點融會貫通，開發一款支援模組機制的完整小型遊戲。專案題材建議為一個極簡版的 Vampire Survivors 類遊戲：玩家操控角色對抗不斷湧現的敵人，獲取經驗升級並撐到最後。核心玩法可以很簡單，但我們要求**從架構上設計良好，以便擴充**。也就是說，遊戲內的**角色類型**、**武器道具**、**敵人行為**等內容，都應通過我們設計的模組系統來實現動態擴充。最終，我們將編寫至少一個**外部模組**作為示範，添加全新的遊戲內容（例如新增一種可選角色“吸血鬼獵人”以及專屬武器），驗證遊戲能正確載入該模組並將其融入玩法中。這個綜合專案不僅產出一個有趣的遊戲雛形，更檢驗學生是否真正掌握了 RimWorld 模組架構的精髓和應用能力。

主要功能模組設計：

- **核心遊戲邏輯：**先開發無模組情況下的核心遊戲——實現基本的玩家角色移動、敵人產生與移動、自動攻擊等機制（可簡化為計時器傷害）。確保核心玩法運轉正常。建議使用Unity 2D環境，因為 Vampire Survivors 本是2D遊戲。**注意：**核心邏輯在架構上要**資料驅動**：例如角色屬性不要寫死，要從可配置資

料讀取；敵人種類、武器列表也用外部定義，以方便後續模組添加。可以將本體遊戲也視作一個“內建模組”，放在類似Core資料夾中。

- **模組系統整合：**將前面各項目的模組機制融入遊戲初始化流程。啟動遊戲時，先載入**核心Defs**（遊戲內建內容），然後掃描`Mods/`資料夾載入所有模組定義、資源、DLL，按照**相依性排序**確定執行順序[15]。對每個模組，載入其XML Def（角色、武器、敵人等Def），合併到遊戲的內容池中；載入其Assemblies並執行Mod類初始化（這裡Mod類可以允許模組在需要時**運用Harmony**補丁本體遊戲的一些邏輯，例如修改經驗計算公式等）。確保**多個模組**同時存在時順利加載且各自內容皆可用，無明顯衝突。
- **可擴充的資料驅動內容：**設計一系列Def來表示遊戲可擴充的元素，例如`CharacterDef`（可玩角色定義，包括名稱、初始屬性、初始武器等）、`WeaponDef`（武器定義，包括傷害、攻擊間隔、子彈類型等）、`EnemyDef`（敵人定義，包括血量、移動速度、行為模式引用等）、`AbilityDef`或`BehaviourDef`（行為/技能定義，用於描述角色或敵人的特殊能力）。**要求：**這些Def結構要允許外部模組新增全新條目。遊戲在運行時根據載入的所有Def來構築內容。例如，遊戲開始時角色選單是根據所有載入的CharacterDef生成列表的，這樣模組添加一個新角色Def就自動出現在選單中。敵人的生成也是從EnemyDef列表隨機抽取，模組新加的敵人也會自然被包含。此部分體現資料驅動的價值：**增加內容而無需改核心代碼**。
- **動態資源與美工：**讓模組能提供自己素材（圖像、音效）。例如模組添加新武器時提供了圖標和子彈圖片，我們的系統應能載入並使用它們。Unity可以透過Addressable或Resources動態載圖。簡化起見可規定素材放Resources同名路徑直接Load。驗證模組素材隨Def一起被應用（如角色貼圖換成模組提供的新形象）。
- **模組設定與選項：**提供一個遊戲內界面列出所有載入的模組，點擊可查看模組資訊（名稱、作者、描述、版本等）以及調整其**Mod Settings**（如果有）。模組設定面板可重用前面項目5的成果，每個模組的Mod類提供DoSettingsWindow，我們在主遊戲做一個選單容器呼叫相應模組繪製UI[57]。確保玩家可以調整模組提供的選項且保存。
- **示範模組製作：**開發至少一個獨立的模組作為成果展示。比如創建“**VampireHunter Mod**”：添加一個新可選角色“獵人”（CharacterDef定義生命值、速度、模型等），一把新武器“十字弓”（WeaponDef，帶特殊屬性），一種新敵人“吸血鬼”（EnemyDef，帶特殊移動模式），以及實現一個敵人對玩家吸血的特殊能力（可能需要一個Harmony Patch改動命中計算，使吸血鬼攻擊時偷取玩家生命）。還可以添加Mod Settings讓玩家調整新角色的某些屬性如攻擊力倍率。把這個模組打包在Mods資料夾，用我們的系統載入驗證：遊戲啟動後應該在選角介面看到“獵人”，遊戲中會出現吸血鬼敵人，他們攻擊有吸血效果（透過模組代碼Patch實現），玩家也可在模組設定裡調節相關參數。這完整地演示了我們模組系統的**擴充能力**。
- **文件與指引：**撰寫模組系統使用說明（給未來模組開發者）。包括：如何建立正確的模組目錄、各種Def XML格式說明、如何編寫模組DLL（特別是Mod主類如何繼承或標記）、可用的Hook點（比如我們允許在Mod類的某些虛方法實現功能）、還有如何聲明依賴等等。這相當於我們這個模組系統的“Mod開發指南”，類似RimWorld官方給模組作者的文檔。對學生而言，寫這份指南的過程也是對系統全貌的最後檢驗。若有疏漏之處會在整理文檔時暴露，便於回頭補強。

開發建議：

1. **分階段實現，逐步集成：**由於此專案整合了前面各種功能，開發時應採用增量方式。一開始專注**核心遊戲玩法**，確保有一個可玩的原型。接著逐一融入模組支援：先把資料定義換成外部XML載入（等同於項目1效果），再加入多模組目錄掃描（項目2），然後動態載入模組DLL（項目3），再允許模組Harmony Patch（項目4），然後提供設定介面（項目5），最後處理模組依賴順序（項目6）。每加一部分都要反覆測試遊戲是否仍正常運作，以及新加的模組功能是否OK。分步完成比一次性上更穩妥。
2. **嚴格遵守模組介面約定：**在本專案中，我們扮演“遊戲開發者”，也相當於RimWorld的Tynan角色，需要制定好我們的模組機制的規範，並**堅持不隨意變動**，確保模組與主程式介面穩定。例如約定Mod類必須繼承`BaseMod`並實作`Initialize()`方法，那我們載入器就按此約定執行。一旦上線流通，就不能輕易改規約，以免破壞相容。這讓學生體會到設計API/框架需要前瞻周全，盡量減少未來改動。
3. **廣泛測試：**模組系統的整合非常容易出現邊緣情況，需要通過大量測試來發現問題。建議編寫多種模組插件作測試用例：比如一個只帶數據不帶DLL的模組、一個帶DLL且做Harmony Patch的模組、兩個互

相依賴的模組、內容衝突的模組等等。逐一測試這些情境，觀察遊戲行為和日誌輸出是否符合預期。尤其注意**模組卸載或順序改變**對存檔的影響（如果時間允許可涉略模組移除時的存檔相容，但這屬高難可選擇略）。

4. **性能與擴充考量**：雖然我們遊戲規模小，但還是要關注模組機制對性能的影響。例如模組很多時初始化是否明顯變慢？可以在Loading畫面增加進度條或模組載入訊息提示，用戶體驗會好很多 [81] [82]。此外考慮未來擴充，如果要支持**熱重載**模組（運行時增減）或**模組更新**，需要特殊處理，但非本專案重點，可在文檔中提及作為未來改進方向。
5. **借鑑成熟項目**：在構建的過程中，多參考RimWorld或其他可模組化遊戲的做法。Kerbal Space Program、Cities: Skylines這些Unity遊戲都有自己的Mod系統 [83]。儘管原理不同，但核心都是讓遊戲讀取外來內容。我們可以汲取他們在用戶體驗上的一些優點，比如**模組管理UI**、**錯誤提示友好**、**盡可能避免讓玩家修改原始檔**等等。

里程碑與交付：

- 第一階段交付：遊戲核心Demo – 無模組下的遊戲可玩Demo，展示基本玩法。
- 第二階段交付：模組機制集成測試 – 實現模組載入與內容擴充，在測試場景中讀入幾個示例模組並證明數據和代碼擴充有效（例如打印模組初始化log，出現模組定義的新物件）。這階段可能只是在Console驗證。
- 第三階段交付：完整遊戲&示範模組 – 最終可運行的遊戲，搭配一到兩個示範模組。提供使用說明文檔，指導如何安裝模組和創作簡單模組。可以錄製一段小影片或GIF，演示在沒有模組和安裝模組後遊戲內容的變化，突出我們系統的效果。

透過該綜合專案，學生將最終體認到：開發一個**支援模組**的遊戲，需要在最初架構設計上就把擴充點考慮周全 [32]。這雖增加開發負擔，但長遠看極大提升了遊戲生命力和社群參與度。他們也會為自己實作出一個“小型RimWorld級”的模組系統而感到成就滿滿。此專案作品在未來無論作為技術展示還是實際遊戲開發經驗，都將非常有價值。

推薦資源與參考：

- **《邊緣世界》模組開發者指南** – RimWorld官方或社群編寫的Mod開發指南文件（如PDF或論壇帖），總結模組系統各方面注意事項。對比我們專案，如發現差異，可思考原因。例如我們的設計可能更簡化，而RimWorld更全面。這有助於查缺補漏。
- **《Creating a Moddable Unity Game》博客系列** – 先前提到的TuriyaWare博客 [84] [85] 及其評論中，有對多模組衝突處理的探討，值得重讀。在我們整合專案中特別相關的是如何**合併多個模組修改**同一內容的問題。RimWorld用了PatchOperations，我們簡化處理成後載優先或直接不允許衝突。無論如何，這篇博客提供了不同解決途徑的權衡，對架構師思維有啟發。
- **GitHub 開源遊戲專案：**參考一些專門為易模組化而設計的開源遊戲，如 Veloren (Rust) 或 GODOT Engine 的模組系統。雖語言引擎不同，但理念相通。特別是Veloren也採用資料驅動+插件程式的結構，閱讀其技術博客可以激發我們對自身專案改進的靈感。
- **Unity Asset Store 工具：Mod Tool/uMod** – 了解現有Unity模組框架插件的功能 [86]。例如ModTool聲稱可直接加載C#腳本作為mod，uMod提供了編輯器整合。雖然我們未使用這些工具，但它們的特性（如支持運行時編譯、提供用戶介面等）值得知道。這些也許能引導我們未來為系統增加更高級功能。
- **社群測試與反饋：**如果有機會，可邀請幾位同學或開發者實際依照我們提供的指南做一個小模組，看看他們的體驗。從中獲得反饋可以發現我們系統難用的地方（例如某步驟說明不清）。這種用戶測試反饋在專案最後非常寶貴，能最終完善整個模組開發支援環境，使之更趨於嚴謹和易用。

以上即為針對 RimWorld 模組開發技術的完整學習路徑圖與教材方案。通過循序漸進的項目實作與最終統合專案，學習者將逐步建立從資料驅動設計、模組封裝與載入、Harmony程式碼修改、到模組設定與依賴管理的全盤知識體系，最終能運用這些技術構建出一個支援模組機制的遊戲雛形，為將來開發大型模組化遊戲打下堅實基礎。

---

1 2 6 16 17 22 23 30 31 〖筆記〗如何讓遊戲支援模組開發 - AngusChan's Devlog
https://angus945.github.io/learn/game-development/how-to-make-modification-games/

3 19 20 21 24 PatchOperations - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/PatchOperations

4 5 11 12 38 39 40 41 42 43 44 45 46 47 48 50 51 53 Modding Tutorials/Harmony - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Harmony

7 8 25 26 27 28 79 Mod Folder Structure - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Mod_Folder_Structure

9 10 13 55 56 57 58 61 63 67 Modding Tutorials/ModSettings - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/ModSettings

14 15 29 64 69 70 71 73 74 75 76 77 78 About.xml - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/About.xml

18 37 Modding Tutorials/Getting started with mods - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Getting_started_with_mods

32 33 72 81 82 83 84 85 86 Creating A Moddable Unity Game
https://www.turiyaware.com/creating-a-moddable-unity-game/

34 35 README.md
https://github.com/Fy-/FyWorld/blob/d4c78ef03587e5b0cb79a9e0c3709b81073ef21d/README.md

36 How to update your Mod from RimWorld from 1.0 to 1.1 ... - GitHub Gist
https://gist.github.com/pardeike/08ff826bf40ee60452f02d85e59f32ff

49 Modding:Harmony - Official Caves of Qud Wiki
https://wiki.cavesofqud.com/wiki/Modding:Harmony

52 Patching - Harmony
https://harmony.pardeike.net/articles/patching.html

54 Tutorial: Harmony Patching - Stardew Modding Wiki
https://stardewmodding.wiki.gg/wiki/Tutorial:_Harmony_Patching

59 60 62 65 66 68 RimWorld Saving For Dummies | RimWorld Modding Resources
https://spdskatr.github.io/RWModdingResources/saving-guide.html

80 How important is mod order? : r/RimWorld - Reddit
https://www.reddit.com/r/RimWorld/comments/s9n3ix/how_important_is_mod_order/