

如何使用這本書

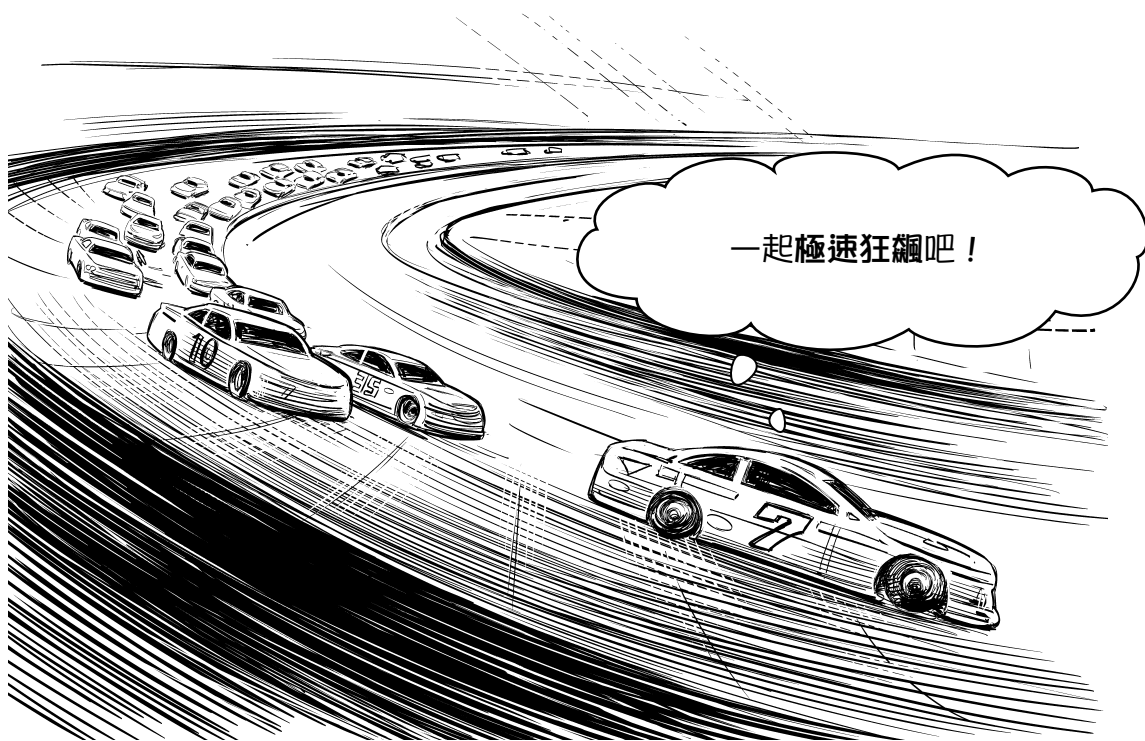
簡介



在這一節，我們要正面迎戰靈魂拷問：「他們到底為什麼要把那些東西放在一本教 C# 的書裡？」。

1 開始用 C# 來建構 app

快速打造出色的作品！



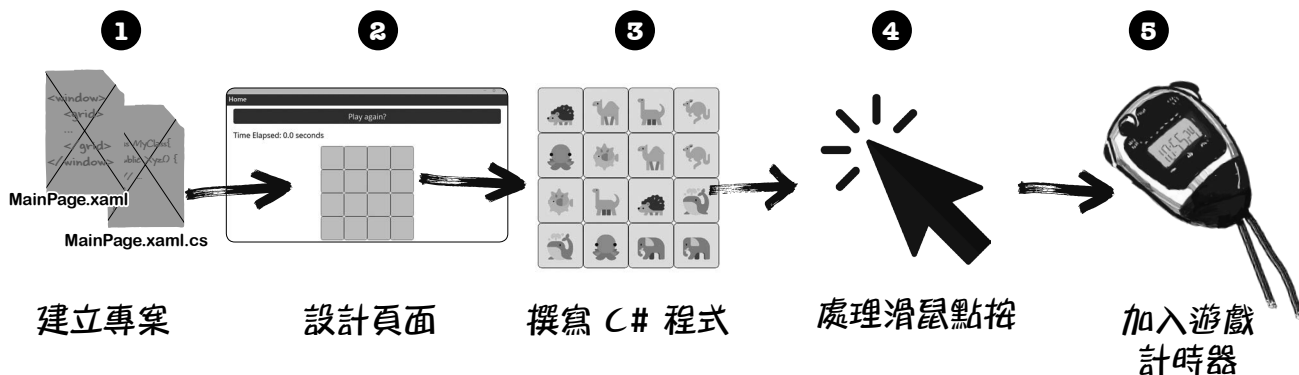
想要寫出偉大的 app 嗎？立刻開始？

學會 C# 就掌握了一種現代程式語言，和一種寶貴的工具。而且學會 Visual Studio 就擁有一種了不起的開發環境，它具備高度直覺的功能，可讓寫程式的過程盡可能地簡單。Visual Studio 不僅非常適合用來寫程式，當你探索 C# 時，也是很有用的學習工具。有沒有吸引你？讓我們開始寫程式吧！

你將如何完成這個專案

將大型專案拆成較小的部分

本書的目標是幫助你學習 C#，並且讓你成為優秀的開發者，而處理大型專案的能力是優秀的開發者必須具備的重要技能之一。在本書中，你將建立許多專案。下一章要啟動的專案最初規模較小，但隨著進度的推進，它們將越來越大。當專案變大時，我們會教你如何將它拆成較小的部分，讓你可以一一完成它們。這個專案也不例外。它是一個較大的專案，類似稍後將進行的專案，因此，你要將它分成五個部分來完成：

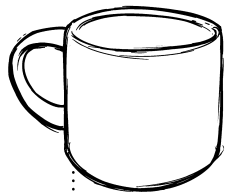


這個專案的目標是幫助你熟悉 C# 的寫法，以及 IDE 的用法。如果你在這個專案中遇到任何問題，可以前往我們的 YouTube 頻道觀看完整的影片教學：

<https://www.youtube.com/@headfirstcsharp>

你也可以從我們的 GitHub 網頁下載所有程式碼與本章的 PDF：

<https://github.com/head-first-csharp/fifth-edition>



放輕鬆

本章的重點是學習基本概念、熟悉專案的建立方法、編輯程式碼，以及瞭解開發遊戲的過程。

如果你有不太理解的部分，不用擔心。到了本書的最後，你將完全掌握這款遊戲的所有原理。現在你只要按照指示，逐步讓遊戲順利運行就好了，這將為後續的學習奠定厚實的基礎。

這就是建構遊戲的方法

你將使用 **.NET MAUI**（.NET Multi-platform App UI 的簡寫，也可以直接稱為 **MAUI**）來建立這款動物配對遊戲。MAUI 這項技術可以讓你使用 C# 來建立 app，這些 app 可以當成桌面應用程式在 Windows 和 macOS 上運行，或當成行動應用程式在 Android 和 iOS 設備上運行。

本章的其餘部分將逐步指導你完成這款遊戲。你將透過以下幾個步驟來完成：

1 首先，在 Visual Studio 中建立新的 .NET MAUI 專案。

你剛剛建立了一個 console app（主控台 app），現在你要建立一個 MAUI app。

2 使用 XAML 來設計頁面。

在 MAUI app 中，每一個獨立的畫面稱為頁面（page）。你將使用 XAML 來設計這些頁面，XAML 是一種用來定義頁面行為的設計語言。

3 使用 C# 程式碼來將隨機的動物 emoji 加入頁面。

當 app 啟動時，它將執行這段程式碼來顯示 16 個按鈕，在按鈕上有隨機排列的 8 對動物 emoji。

4 讓遊戲可以運行。

遊戲必須認知使用者點選了成對的 emoji，記錄已配對的組合，並在使用者找到所有配對時結束遊戲。你也要撰寫這部分的程式碼。

5 最後，加入計時器來讓遊戲更刺激。

計時器會在玩家開始遊戲時啟動，並記錄他們花了多少時間來找到 8 對動物。

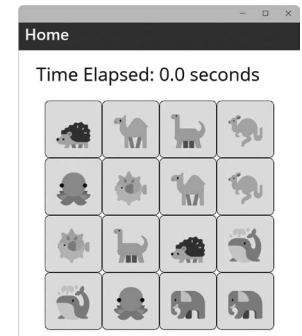
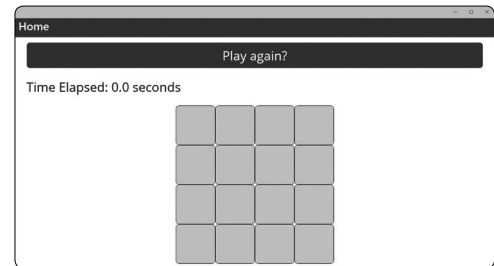
這個專案可能耗時 20 分鐘到 1 小時，取決於你的打字速度。慢慢學習的效果比較好，所以請給自己充足的時間。

```
<FlexLayout>
  <Button/>
  ...
  <Button/>
</FlexLayout>
```

MainPage.xaml

```
class MyClass {
  void Xyz() {
    // ...
  }
}
```

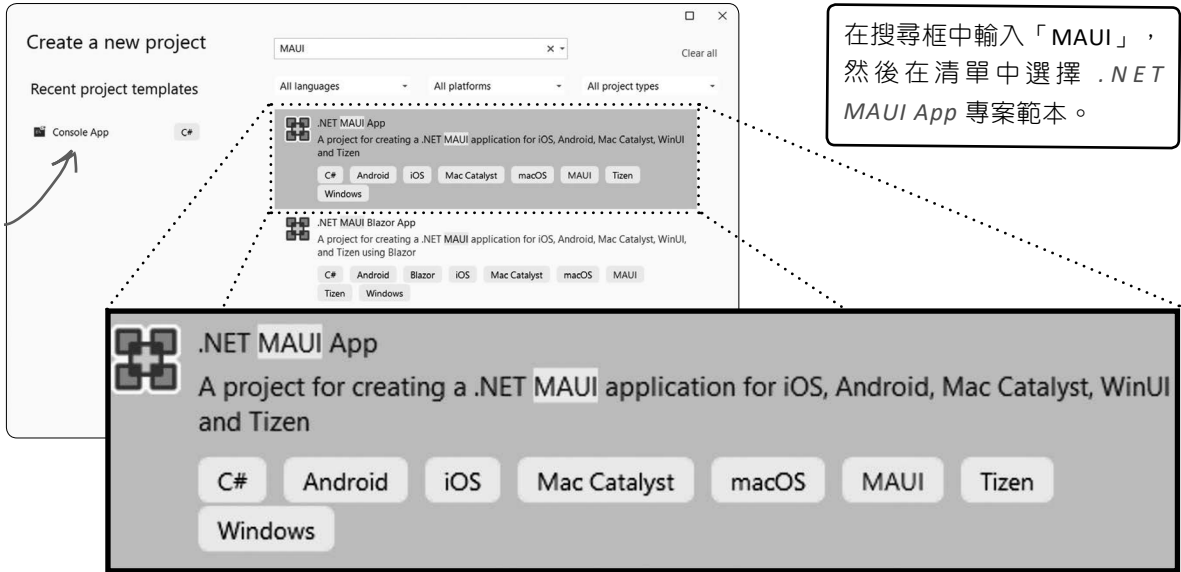
MainPage.xaml.cs



在 Visual Studio 中建立 .NET MAUI 專案

你可以像本章開頭建立 Console App 時一樣，在 Visual Studio 中建立 .NET MAUI app，方法是在第一次開啟 Visual Studio 時，按下「Create a new project」按鈕。如果 Visual Studio 已經開啟，可在 File 選單中選擇 New >> Project（或按下 Ctrl+Shift+N），來開啟「Create a new project」視窗。

你會看到
Console App
範本出現在
最近使用的
範本清單中，
因為你之前
已經建立過
一個了。



選擇 .NET MAUI App 專案範本，然後按下 Next。Visual Studio 會要求你輸入專案名稱，就像你建立 Console App 專案時一樣。

輸入 **AnimalMatchingGame** 作為專案名稱，然後按下 Next。



最後，Visual Studio 會要求你選擇 .NET 版本，請選擇最新的版本，就像你建立 Console App 時那樣。然後按下 Create 按鈕來建立你的 .NET MAUI 專案。

先再三思考，再開始撰寫程式

你在這裡

我們會在這個專案的每一個部分的開頭提供類似這樣的「地圖」，以協助你掌握整體概念。

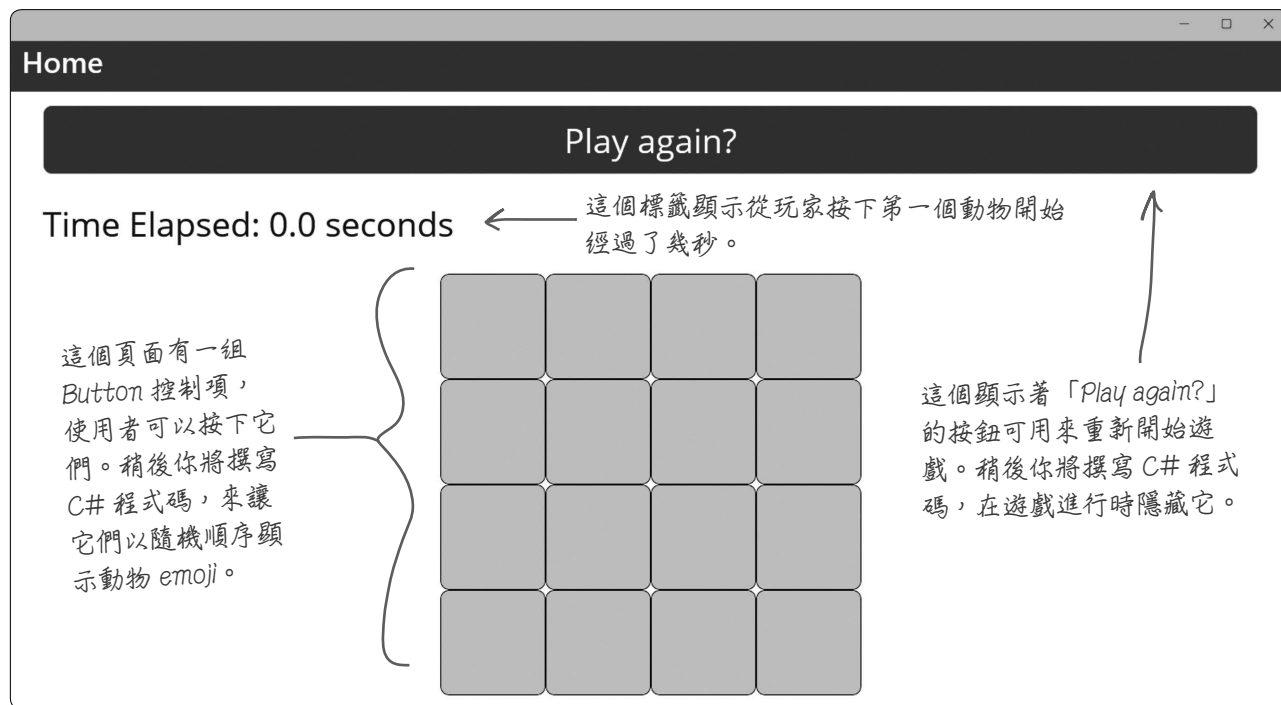


這是你將建立的頁面

當你開始專案時，第一件事就是花幾分鐘來理解整體概念。你要做出什麼？它將如何運作？我們來看看你即將設計的頁面。

當你開啟一個使用 .NET MAUI 來建立的 app 時，首先會看到一個可以互動的頁面。這個頁面使用**控制項 (control)** 或視覺 widget (例如按鈕與標籤) 來建立一個可以互動的使用者介面 (UI)。以下是你即將設計的頁面：

你將使用 XAML 來設計頁面。許多 C# 開發者認為 XAML 是一項核心技能，不少 C# 工作職位都需要基本的 XAML 知識，因此我們希望在本書中加入足夠的 XAML 內容，為你建立扎實的基礎。



開始編輯你的 XAML 程式碼

在 Solution Explorer 中，你可以按兩下程式碼檔案來編輯它們（在 VSCode 中則是按一下）。我們將使用兩個檔案：*MainPage.xaml*（包含 XAML 程式碼）和 *MainPage.xaml.cs*（包含遊戲的 C# 程式碼）。以下是在 Visual Studio 中的畫面：

如果你關閉了任何一個檔案，你可以在 Solution Explorer 中對著它們按兩下來重新開啟它們。如果看不到 Solution Explorer，你可以在 View 選單中選擇 Solution Explorer 來開啟它。

你可以使用這些頁籤來切換專案中的檔案。

VSCode 的外觀略有不同，但頁籤的動作相同。你只要在 Explorer 面板中展開 *Solution Explorer* 即可。將最上面的區塊折疊起來對你會有幫助。

務必在 Solution Explorer 裡展開 *MainPage.xaml*，這是查看 *MainPage.xaml.cs* 的不二法門。

你將在這個專案中編輯這兩個檔案。



小心！

Visual Studio Code 為 XAML 程式碼的編輯提供的支援有限。

當你使用 Visual Studio Code 來編輯 *MainPage.xaml* 裡的 XAML 程式碼時，可能會發現它缺少本章提到的一些功能。當你在 Visual Studio 裡編輯 XAML 程式碼時，它提供以下這些方便的功能：

- ★ Visual Studio 有一個 Toolbox，裡面有可以拖曳到 XAML 編輯器裡的控制項，也有 Properties 視窗，讓你能夠輕鬆地編輯控制項的屬性。
- ★ 當你新增事件時，Visual Studio 會自動建立事件處理方法的程式碼：

```
FontSize="Large" Clicked="|" />
```

<New Event Handler>

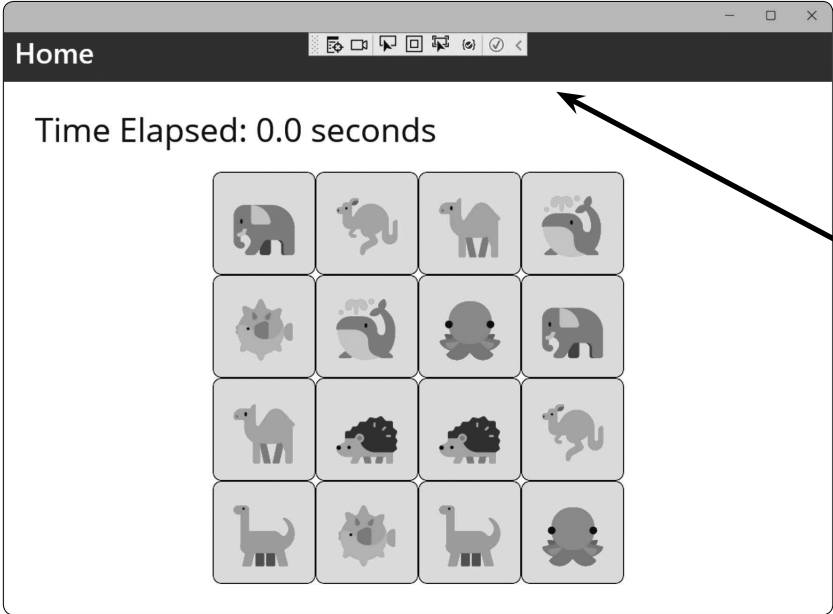
Visual Studio 會幫助你將事件連接到事件處理方法，或建立新的事件處理方法。

在筆者撰稿時，VSCode 的 .NET MAUI 擴充功能尚未包含這些功能，因此你要親自撰寫程式碼，無法讓 IDE 自動幫你產生它們。

你做出它了，而且它可以正確運作

執行你的 app ！

再次執行 app。首先，你會看到「Play again?」按鈕。按下按鈕應該會顯示八組隨機排列的動物 emoji：



停止執行 app，然後多次重新執行它。每次按下「Play again?」按鈕時，動物的順序都會重新隨機排列。

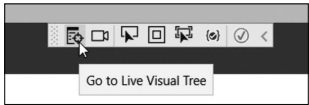


哇！這個遊戲有點樣子了！

你已經做好加入下一個部分的準備了。

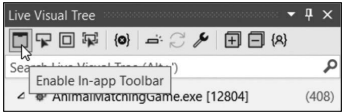
建構新遊戲不是只要寫程式就好了，你也要執行專案。有一種高效率的專案管理方法是將它拆成小步驟來開發，並在過程中評估進展，以確保一切都朝著正確的方向前進。如此一來，你會有足夠的機會根據需求調整方向。

如果你使用的是 Visual Studio，你可能會在視窗的最上面看到 app 內部工具列：



我們在螢幕截圖中隱藏了 app 內部工具列。你可以讓它保持顯示，或使用右側的箭頭來將它摺疊。

想要的話，你也可以關閉它（但不一定需要如此！）。按下 runtime 工具的第一個按鈕會在 IDE 中顯示 Live Visual Tree 面板：



然後按下 Live Visual Tree 中的第一個按鈕，以開啟或關閉 app 內部工具列。

這是一個紙筆練習。我們在這本書中加入許多這樣的遊戲和謎題。請完成所有練習，因為來自神經科學的證據指出，書寫可以將重要的概念迅速植入大腦。



連連看?

C# 陳述式

```
List<string> animalEmoji = [
    "🐼", "🐼",
    "🐼", "🐼",
    "🐼", "🐼",
    "🐼", "🐼",
    "🐼", "🐼",
    "🐼", "🐼",
    "🐼", "🐼",
    "🐼", "🐼",
    "🐼", "🐼",
];
```

它做了什麼

讓按鈕顯示所選的 emoji

在 FlexLayout 中找到每一個按鈕，
並為它們執行大括號 { } 之間的陳述式

隱藏「Play again?」按鈕

```
foreach (var button in AnimalButtons.Children.OfType<Button>())
{
```

建立一個包含八對 emoji 的串列

```
    animalEmoji.RemoveAt(index);
```

讓包含 emoji 按鈕的 FlexLayout 顯示出來

```
    button.Text = nextEmoji;
```

```
    string nextEmoji = animalEmoji[index];
```

選擇一個介於 0 和串列剩餘的 emoji 數量
之間的隨機數，並將它命名為「index」

```
    AnimalButtons.IsVisible = true;
```

將被選中的 emoji 從串列中移除

```
    int index = Random.Shared.Next(animalEmoji.Count);
```

```
    PlayAgainButton.IsVisible = false;
```

使用名為「index」的隨機數從
串列中隨機取出一個 emoji



遊戲設計…和雜談

機制

遊戲**機制**是構成實際玩法的遊戲元素：遊戲規則、玩家可以採取的行動，以及遊戲如何回應玩家的行動。

- 我們從一個經典的遊戲說起。**小精靈的機制**包括如何用搖桿來控制螢幕畫面中的主角、小圓點和大力丸的點數、鬼魂如何移動、它們變成藍色的時間、當主角吃到大力丸時，鬼魂的行為如何改變、主角何時可以獲得更多條命、鬼魂經過隧道時如何減速——所有的規則驅動了整個遊戲。

如果你沒有玩過小精靈，請花幾分鐘看一下別人玩它的影片，最好可以自己玩一下！你可以透過很多管道玩到這款遊戲，最簡單的一種是在 Google 搜尋「Pac-Man」。Google 為他們的 30 週年建立了一個可遊玩的小精靈遊戲，可用箭頭按鍵在瀏覽器裡遊玩。

- 遊戲設計師口中的**機制**通常是一種互動或控制模式，例如平台遊戲的二段跳，或射擊遊戲中只能承受數次砲擊的護盾。將機制獨立出來測試與改善通常很有幫助。
- **桌面遊戲**可以讓我們充分瞭解機制的概念。骰子、轉盤或卡牌等產生亂數的手段都是機制的典型案例。
- 你已經看過一個很好的機制案例了：在動物配對遊戲中的**計時器**可以改變整個遊戲的體驗。計時器、障礙物、敵人、地圖、種族、點數…都是機制。
- 不同機制之間有各種不同的**組合**，它們一起深刻地影響遊戲的體驗。大富翁就是一個很好的例子，它結合兩種不同的隨機數產生機制（骰子和卡牌），讓遊戲更有趣且更微妙。
- 遊戲機制也包含**資料如何架構，以及處理資料的程式如何設計**，即使該機制是無意造成的！小精靈的傳奇關卡 **256 glitch** 有一個 bug 會用垃圾資訊顯示一半畫面，導致遊戲無法遊玩，它也是遊戲機制的一部分。
- 使用 C# 寫出來的遊戲的機制**包括類別與程式碼**，因為它們驅動遊戲的運作方式。

這是另一個「遊戲設計…和雜談」的專欄。這次的主題是機制，它是電子遊戲設計的重要成分。即使你沒興趣撰寫電子遊戲，也一定要閱讀這些專欄！它們有很多重要的概念，而且我們將在本書的其餘內容中延伸它們。



我猜，機制的概念可以幫我進行任何一種專案，而不是只有遊戲而已吧？

確實如此！每一個程式都有它自己的機制類型。

每一個軟體設計層面都有機制。在遊戲背景之下，我們比較容易討論和理解機制。我們將透過遊戲來協助你更深入瞭解機制，對於設計和建構任何一種專案來說，機制都很有價值。

遊戲的機制決定了它玩起來多難或多簡單。舉一個例子：讓小精靈跑得更快，或是讓鬼跑得更慢，都會讓遊戲更簡單。這不一定會讓遊戲變得更好或更壞，只是會帶來改變。猜猜怎麼著？同樣的概念也適用於類別的設計！你可以將**如何設計方法與欄位**視為類別的機制。如何將程式碼拆成方法，或使用欄位的時機，將會影響程式碼更容易或更難以使用。

重點提示

- **方法**由陳述式組成。呼叫方法會依序執行它的陳述式。
- 將陳述式放入方法中並為方法命名可以讓程式碼更易讀。
- 當方法執行完所有陳述式或執行 **return** 陳述式時，**執行流程**會回到當初呼叫方法的**陳述式**之後繼續執行。
- 變數的**型態**決定了它可以儲存的資料型態，例如整數、小數、文字，或 **true/false** 值。
- 在使用變數之前，必須先幫變數**設值**。
- **運算子**（如 **+**、**-**、***** 和 **/**）可以操作變數儲存的資料。**=** 運算子可指派值，而 **==** 運算子可比較兩個值是否相等。
- **if 陳述式**可讓程式在你設定的某些條件為 **true**（或為 **false**）時執行特定的動作。
- **迴圈**會重複執行一組陳述式，直到特定條件滿足為止。**for**、**while** 和 **do/while** 迴圈都會多次執行陳述式，但它們的運作方式各有不同。
- Visual Studio 的**程式碼 snippet** 功能可以幫助你撰寫 **if** 陳述式和迴圈。



第一課：AI 聊天機器人並不完美…但沒關係

當你使用 Copilot、ChatGPT 或 Gemini 等 AI 聊天機器人時，你會先輸入一個**提示詞**，或問題、敘述，或其他類型的查詢句，以開啟對話。我們使用以下的提示詞來生成本節的介紹：

I'm the author of Head First C#, a book that aims to help people learn about development with C# in a unique and engaging way. A section of that book is about how to use AI chatbots as a tool to learn and explore C# on their own. Can you help me write the introduction to that section?

AI 產生了一篇很棒的介紹——好到讓我們將之收錄於書中！但它並不完美。看看第二句，「身為一名作者，你…（As an author, you've...）」——這不太對。聊天機器人可能因為提示詞的開頭是「I'm the author（我是…的作者）」而有些困惑。這本書的對象不是作者，而是正在學習 C# 開發的人。你的使命不是讓學習變得有趣且難忘——那是我們的使命！

AI 能夠生成這麼棒的内容真的很驚人，即使我們可能想要稍微修改一下（例如把「pedagogical（教育）」換成「teaching-related（教學相關的）」）。

如何安全地使用 AI：信任，但要驗證

對開發者而言，使用 AI 聊天機器人是越來越重要的技能。我們的目標之一是幫助你**安全地**使用 AI 聊天機器人，這意味著你必須**自行確認**兩件事：你從 AI 學到的知識是否都**正確**，以及 AI 為你產生的程式碼是否真的**做你想做的事情**。我們將在本書中加入這些 Sens-AI 單元，以幫助你學會如何確認這些資訊，並安全地使用 AI。

以下是 AI 的優勢與缺點案例。我們將這個提示詞傳給 Copilot、ChatGPT 和 Gemini：

I have the following for loop in C#:

```
int p = 2;
for (int q = 2; q < 32; q = q * 2)
{
    while (p < q)
    {
        // 下一個陳述式
        // 會執行多少次？
        p = p * 2;
    }
    q = p - q;
}
```

我們直接從本章稍早的「削尖你的鉛筆」部分複製了這段程式碼。你可以直接在提示詞中加入程式碼。

當你在瀏覽器裡將提示詞傳給 AI 聊天機器人時，可能需要使用 **Shift-Enter** 來換行。我們建議使用文字編輯器來編輯提示詞，例如 VSCode、Windows Notepad 或 macOS TextEdit。然後將它們複製並貼入瀏覽器中，如此一來，你也可以將它們儲存在資料夾中，以供日後使用。

How does it work, and how many times are the inner and outer loops executed?

AI 聊天機器人仔細地解釋了這段迴圈如何運作，但是這三種聊天機器人都**答錯了**迴圈的執行次數。事實上，它們三個給出彼此不同的答案。我們在幾天之後又問了同一個問題，結果三個聊天機器人又各自給出完全不同的錯誤答案。但沒關係！AI 技術還在持續進步，也許它不會有完美的一天，但它始終是一項**有用、寶貴的學習工具**。

試試看

AI 研究者還在不斷使用問題及其正確答案來更新模型，因此，AI 可能因為問題和答案已被收錄於其資料庫中，而能夠正確地回答問題。

打開 Copilot (<https://copilot.microsoft.com>)、ChatGPT (<https://chat.openai.com>)、Gemini (<https://gemini.google.com>) 或任何其他 AI 聊天機器人，輸入上面的那段關於迴圈的提示詞。修改提示詞中的程式碼，將它換成「削尖你的鉛筆」練習中的其他迴圈。AI 能否清楚解釋這些迴圈？它算出來的執行次數是否正確？

AI 聊天機器人不一定能夠提供正確的答案，部分的原因是它們使用統計模型來產生句子，而不是真的在思考！但如果你瞭解它們的限制，並練習使用它們，它們將成為非常有用的工具。

Unity 實驗室 #1

用 Unity 來探索 C#

歡迎光臨你的第一個深入淺出 C# Unity 實驗室。寫程式是一門技術，如同任何其他技術，你必須透過實際操作和實驗才能熟練這門技術，而 Unity 是非常寶貴的實驗工具。

Unity 是一種跨平台遊戲開發工具，你可以用它來製作專業的遊戲、模擬程式…等。它也是非常適合用來練習本書將教導的 C# 工具和想法的有趣管道。我們設計這些簡短、有目標的實驗來**強化你學到的概念與技術**，協助你磨練 C# 的技能。雖然這些實驗是可省略的，但它們是寶貴的練習，**即使你不打算使用 C# 來製作遊戲**。

在第一個實驗室中，我們將教你使用 Unity。你將會熟悉 Unity 編輯器，並開始製作和操作 3D 形狀。這將為你奠定基礎，讓你在下一個實驗中開始撰寫程式。

Unity 是強大的遊戲設計工具

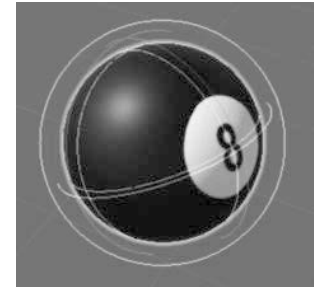
歡迎光臨 Unity 的世界，它不僅是完整的專業遊戲設計系統（包括二維（2D）與三維（3D）遊戲），也可以用來設計模擬器、工具和各種專案。Unity 有許多強大的工具，包括…

跨平台遊戲引擎

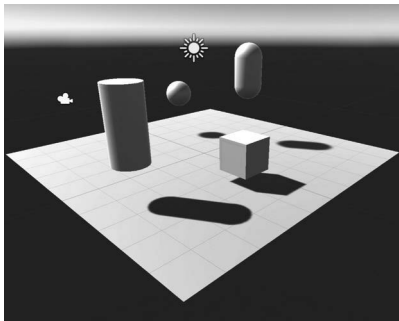
遊戲引擎可以顯示圖形、追蹤 2D 或 3D 角色、檢測它們何時撞在一起、讓它們像真實物體一樣動作，以及許多其他功能。Unity 可以為你在本書中建構的 3D 遊戲裡執行以上的所有事情。

強大的 2D 與 3D 場景編輯器

你將會花很多時間來使用 Unity 編輯器。它可讓你編輯充滿 2D 或 3D 物件的關卡，提供許多工具來讓你設計完整的遊戲世界。Unity 遊戲使用 C# 來定義物件的行為，Unity 編輯器可以和 Visual Studio 整合，提供無縫的遊戲開發環境。



雖然這些 Unity 實驗室的主題是在 Unity 裡進行 C# 開發，但如果你是視覺藝術家或設計師，Unity 編輯器也有很多專門為你設計的藝術家工具。詳情請參考：
<https://unity.com/solutions/artist-designers>。



遊戲製作生態系統

Unity 不但是強大的遊戲製作工具，也擁有一個可以協助你創作和學習的生態系統。Learn Unity 網站 (<https://unity.com/learn>) 有寶貴的自主學習資源，Unity 論壇 (<https://forum.unity.com>) 可協助你認識其他的遊戲設計師並請教他們。Unity Asset Store (<https://assetstore.unity.com>) 有免費與付費的資源，例如角色、形狀與效果，可在你的 Unity 專案中使用。

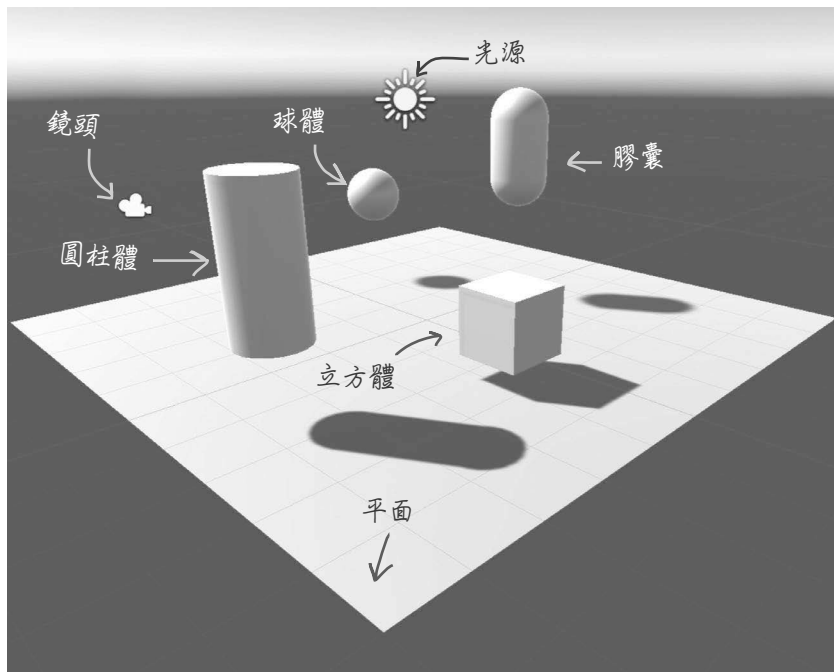
Unity 實驗室將 Unity 當成探索 C# 的工具，在裡面練習 C# 工具和本書教導的概念。

深入淺出 C# Unity 實驗室將焦點放在以開發者為中心的學習途徑上。這些實驗室的目標是協助你快速掌握 Unity，它們採用與深入淺出 C# 相同的、適合大腦的即時學習方案，提供具有針對性的大量練習來讓你有效掌握 C# 概念和技術。

Unity 遊戲是用 GameObject 來製作的

在場景中加入球體就會製作一個新的 **GameObject**。GameObject 是 Unity 的基本概念。在 Unity 遊戲裡面的每一個項目、形狀、角色、光線、鏡頭，或特效都是一個 GameObject。你在遊戲中使用的任何場景、角色和道具都是用 GameObject 來表示的。

在這些 Unity 實驗室裡，你將用各種不同的 GameObject 來建構遊戲，包括：



GameObject 是 Unity 的基本物件，元件是其行為的基本元素。Inspector 視窗會顯示場景中的各個 GameObject 的細節及其元件。

每一個 GameObject 都有一些元件（**component**），提供它的形狀、位置，並且賦予它的所有行為。舉例來說：

- ★ 轉換元件（*transform component*）決定 GameObject 的位置與轉向。
- ★ 材質元件（*material component*）藉著改變顏色、反射、光滑度…等，來改變 GameObject 的轉譯（**render**）方式（也就是 Unity 繪製它的方式）。
- ★ 腳本元件（*script component*）使用 C# 腳本來決定 GameObject 的行為。

ren-der（轉譯），動詞。

以藝術方式呈現或描繪。米開朗基羅在呈現他最喜歡的模特兒時，使用了他在任何其他畫作都未曾使用的細節。

譯註：render 在不同的語境、不同的軟體有不同的譯法，微軟官方譯為「轉譯」，Unity 目前譯為「渲染」，此外還有「算繪」、「彩現」、「設色」等譯法。無論如何，在左側的句子裡，說米開朗基羅在「轉譯」或「渲染」他的畫作都很奇怪，所以只好譯為「呈現」。雖然譯者認為將 render 譯為「算繪」、「設色」都不錯，但本書一律採用微軟的官方譯法：轉譯。

類別可協助你組織程式碼

坦白說…你將在這本書裡撰寫大量的程式碼。而且，隨著你繼續學習，專案將會越來越大，這是好事！

較大的 app 會帶來一些有趣的挑戰。你在第 2 章結束時建立的 app 只有少量的方法。當你建立一個方法一樣多的 console app 時，完全沒有不將方法全都放在 Program.cs 裡的理由。

但是到了書的結尾，你將建立擁有數十個方法的 app。如果把所有方法都放到同一個巨大的 Program.cs 檔案中，你將難以記住每一個方法的用途，而且，如果有難以捉摸的 bug，它一定會把你逼瘋！

幸運的是，C# 為這種挑戰提供了一種解方。你的 C# 程式碼是由**類別**組成的，這些類別包含方法。你當然可以將所有方法都放入一個大類別裡，而一些小型的 app 可能只需要一個類別。但是當你的程式碼變多時，**根據功能來組織類別**更加合理。如果你將類別整理得很直覺，你就可以輕鬆地知道該在哪裡新增方法，同時，也讓 debug 變得更簡單。



解剖 C # App



每一個 C# 程式的程式碼都是用一模一樣的模式來建構的。所有的程式都會使用名稱空間、類別與方法來讓程式碼更容易管理。

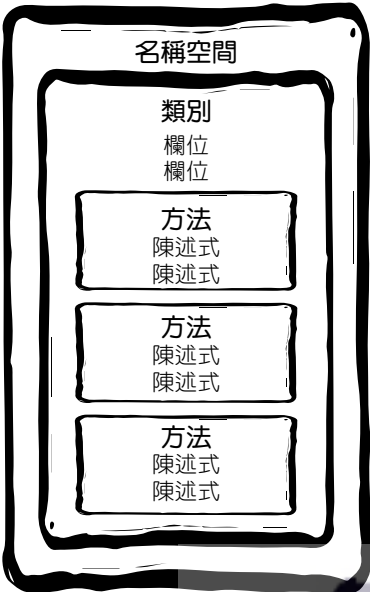
當你建立 app 時，所有的程式碼都位於一個名稱空間內。這可以將你的類別與 .NET 提供的類別分開。

類別包含整體程式的一部分。有一些非常小型的程式只有一個類別，但大多數程式都有多個類別。

類別可以擁有欄位。欄位是一種變數，但它是在方法之外宣告的，所以該類別的所有方法都可以使用它。

一個類別有一或多個方法。所有方法都必須位於類別之內。方法在類別裡的順序無關緊要。方法 2 也可以放在方法 1 前面。

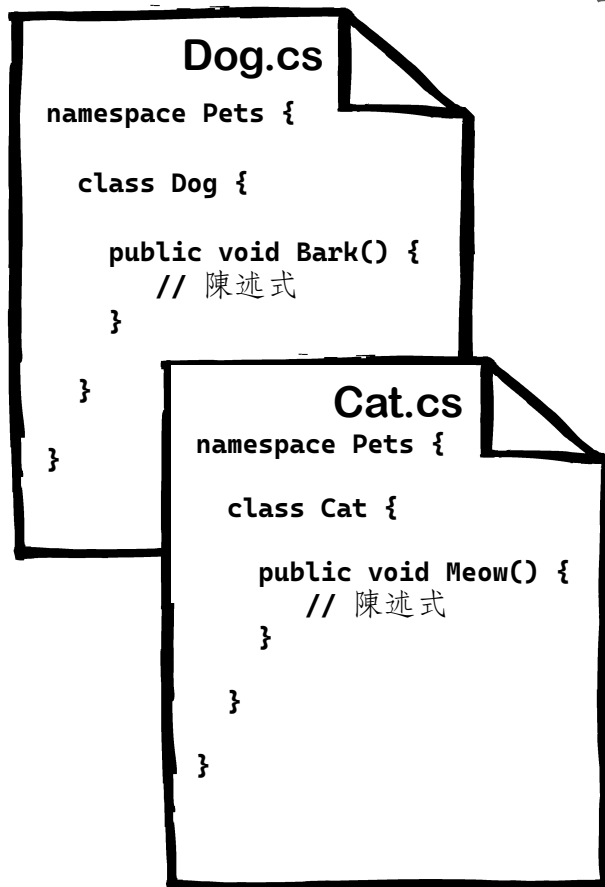
方法由陳述式組成——就像你在前兩章的 app 中使用的那些陳述式一樣。



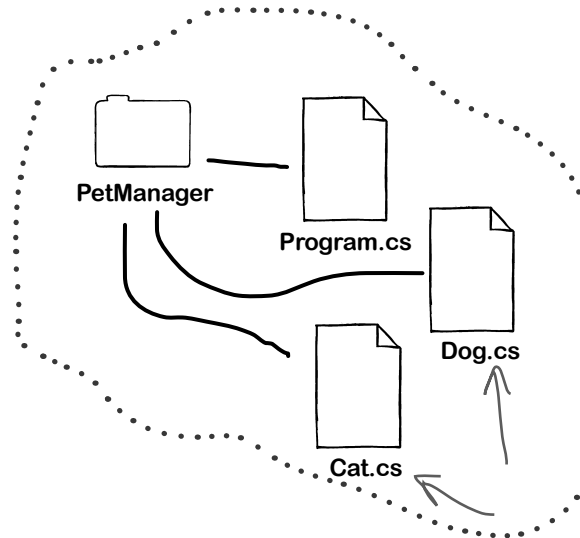
如果程式碼很有用，類別可以幫助你重複使用它

開發者從程式設計的早期階段就開始重複使用程式碼了，原因並不難猜。如果你曾經為 A 程式寫過一個類別，後來在 B 程式中需要加入做同一件事的程式碼，那麼在 B 程式中重複使用同一個類別是很合理的做法。因此，如果要建立一個名為 PetManager 的 app，我們可能會使用名為 Dog 和 Cat 的類別來組織程式碼。

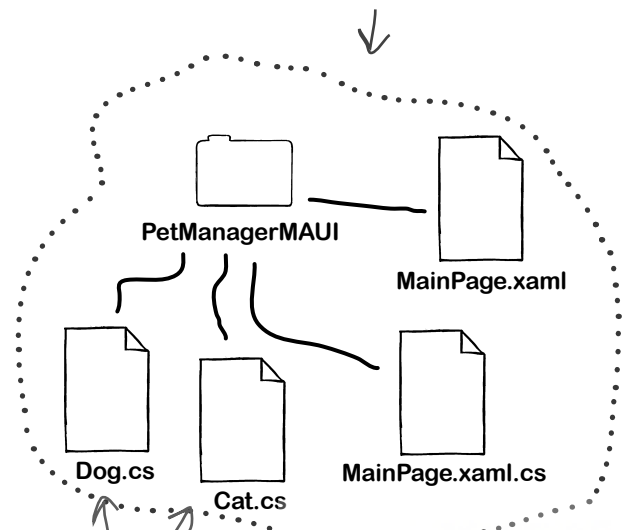
我們為 PetManagerApp
console app 製作了 Dog
與 Cat 類別...



由於我們將類別放入 Pets 名稱空間，因此只要將檔案複製到新專案，並在使用 Dog 或 Cat 類別的任何一個 C# 程式碼檔案的最上面加上「using Pets;」。



...後來發現，在 MAUI 版本的 PetManager app 中也需要使用完全相同的類別，所以我們重複使用它們。



這些是存有 Dog 和 Cat 類別
的同一對檔案。