Switch Mode: 探討過往程式設計經驗對於混合程式語言環境的影響

Switch Mode: Understanding the Effects of Prior Programming Experiences in a Hybrid Programming Environment

林榆涵 [1], David Weintrop [2], Jason McKenna [3], 李大安 [4], 羅木迪 [5]
Yuhan Lin[1], David Weintrop [2], Jason McKenna [3], Andy Lee [4], Mudi Luo [5]
[1] VEX 機器人公司 計算機科學教育主任
[1] VEX Robotics Director of Computer Science Education
E-mail: jimmy@vex.com
[2] 馬里蘭大學 教育與學習暨政策與領導力研究 教授
[2] University of Maryland Department of Teaching and Learning, Policy and Leadership Assistant Professor
E-mail: weintrop@umd.edu
[3] VEX 機器人公司 全球教育策略副總裁
[3] VEX Robotics VP of Global Educational Strategy
E-mail: jason@vex.com
[4] IFI(香港)有限公司 總裁
[4] IFI (HK) Ltd President
E-mail: andy_lee@innovationfirst.com
[5] 淡江大學 教育科技系 碩士生
[5] Tamkang University Department of Educational Technology Master Student
E-mail: 607734018@gms.tku.edu.tw

摘要

本研究中探討了兩位學習者在混合程式語言環境中學習程式設計的不同學習路徑。混合程式語言環境的設計目的是幫助學習者從視覺化程式語言過渡到文字式程式語言。儘管兩位學習者在過去的程式設計經驗和學習程式設計的自信程度上有所不同，但他們都能夠利用混合程式設計環境提供的支持，發展出適合自己的程式指令編寫方法，從而達到對程式設計感到舒適的程度。本研究探討了同一環境下如何支持學習者發展出適合他們的學習自信程度、偏好和先前經驗的不同程式編寫方法。

關鍵字： 程式語言的環境設計；教育機器人；視覺化程式語言；Switch Mode；文字式程式語言

Abstract

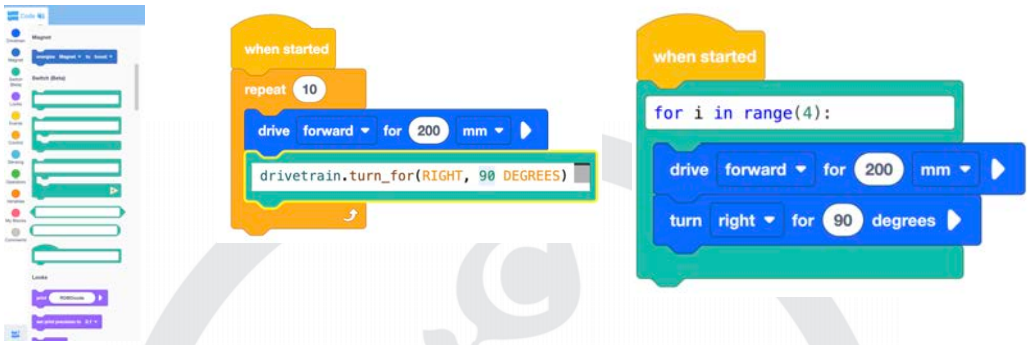
In this paper, we explored the distinct learning trajectories of two individuals navigating a hybrid programming environment, a platform aimed at facilitating the transition from

block-based to text-based programming for learners. With the wide range of previous programming experience and confidence, both students found their level of comfort with programming by developing their own methods for authoring commands based on the supports available in the environment. This work explores how the same environment supported learners in developing distinct programming approaches suited to their confidence, preference, and previous experiences.

Keywords: Design of Programming Environments; Educational Robotics; Block-based programming; Switch mode; Text-based programming

壹、緒論

視覺化程式語言環境是一種有效向初學者介紹程式設計的方式(Weintrop, 2019)。視覺化程式語言環境具有廣泛的理論框架與類型，而且視覺化程式語言設計工具的功能以及能夠使用視覺化程式語言編寫的程式類型正在迅速拓展(Lin & Weintrop, 2021)。目前越來越多的視覺化程式語言環境可支援學習者從視覺化程式語言過渡到文字式程式語言，例如 MakeCode Micro:bit、Pencil Code 和 VEXcode (Lin & Weintrop, 2021)。其中一種新興的過渡方式是在視覺化程式積木內嵌入迷你文本編輯器，它是一種在單一介面或程式中融合兩種程式語言的方式。本研究使用的 Switch Mode（圖 1）正是採用了這種過渡方式。



(a) (b) (c)

圖 1：(a)Switch Mode 在積木選擇區域，(b)單行 Switch Mode 積木 和 (c) C 形狀 Switch Mode 積木

本研究提出的案例研究來自一個大型研究項目，研究項目致力於探討 Switch Mode 面對不同類型學習者其學習方式有何不同，包括想要過渡到 Python 的學習者以及程式語言學習自信心較低或經驗不足的學習者。本研究深入探討學習者的程式設計能力自我認知如何影響他們理解從視覺化程式語言過渡到文字式程式語言的過渡過程。此外，本研究進一步闡述了混合程式語言環境如何設計來支持擁有不同學習動機、自信程度和先備知識的學習者。

貳、背景

視覺化程式語言採用拼圖視覺隱喻作為程式語言指令來減少語法錯誤並消除錯誤信息(Maloney et al., 2010)。學習者透過拖拉程式指令互動來編寫程式碼，按照程式指令的形狀和顏色的視覺提示來引導如何組裝程式。如果兩個程式指令不能結合形成有效序列，視覺化程式語言環境將阻止它們結合，從而在編寫程式碼時避免語法錯誤。視覺化程式語言已被證明能夠有效激勵兒童並使程式語言學習變得簡單 (Moors et al., 2018)。

然而，隨著學習者邁向高中、大學甚至是資訊工程相關職業，他們會需要將自己的程式設計能力從視覺化程式語言轉換為傳統的文字式程式語言，如 Python 或 Java。Lin 和 Weintrop（2021）確定了三種用於支持從視覺化過渡到文字式程式語言的不同設計策略：（1）單向過渡，（2）雙模式，和（3）混合型。在單向過渡環境中，學習者能夠在視覺化程式語言環境中查看或編寫文字式程式，但無法從文字式程式語言返回到視覺化程式語言。雙模式環境則是同時支持視覺化程式語言和文字化程式語言的編寫，允許學習者在兩種模式之間來回切換。而混合型環境是將視覺化程式語言和文字式程式語言的特點結合在同一個介面中，混合程式語言環境的核心是允許學習者在視覺化程式中進行文字編輯。(Kölling et al., 2017)。

由過往研究中發現，與使用文字式程式語言的學習者相比，使用視覺化程式語言的學習者展現出更大的學習成效並且對未來學習資訊工程課程產生更高學習動機（Weintrop & Wilensky, 2017）。另一項研究是在博物館背景下進行觸控式與視覺化程式語言介面應用於程式設計機器人的比較，研究顯示學習者對觸控式介面具有顯著偏好（Horn et al., 2009）。此外，與大學早期的資訊工程中傳統的文字式程式語言的學習者相比，在講座中使用雙模式的學習者在課程考試中表現更佳（Blanchard et al., 2020）。以上三項研究均表明過渡模式的重要性，並且它們是會影響學習者的學習與興趣。研究同時展示學習者如何創造發展出適合不同模式的獨特程式設計方式，這些方式充分利用了介面的特定功能，例如透過積木分類區來激發創意思維，或將視覺化程式指令拖放到文字式程式工作區，以獲得文字式程式語言語法上的理解與幫助(Weintrop & Wilensky, 2018)。

參、研究方法

此研究提出的案例研究來自一個大型研究項目，研究項目致力於探討學習者如何從視覺化程式語言過渡到文字式程式語言。本研究邀請 8 位來自美國中西部城市的高中生參與了一個學期的程式設計課程，使用名為 VEXcode 的混合型視覺化程式語言環境（圖 2）。VEXcode 是一種單向過渡工具，讓學習者可以在虛擬世界中控制虛擬機器人，但是 VEXcode 同時具有讓學習者在視覺化程式積木內編寫 Python 程式碼的功能，這一功能被稱為 Switch Mode（Lin et al., 2023）。Switch Mode 的設計目的是幫助學習者從視覺化程式語言順利過渡到文字式程式語言。

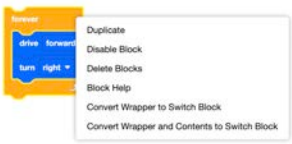
圖 2：VEXcode 程式設計環境

在 Switch Mode 中，學習者有兩種主要的程式語言編寫方法：（1）直接在新的 Switch Mode 積木中輸入文字式程式指令（2）透過右鍵點擊並選擇目錄中提供的轉換功能。Switch Mode 積木的近似於在積木分類區中的傳統視覺化程式積木（圖 1a），不同的部分是增加了學習者可以直接在其中輸入 Python 程式碼的功能。Switch Mode 積木與許多整合開發環境（IDE）系統相同，設置了自動完成系統（圖 3c）。當學習者開始在 Switch Mode 積木內輸入時，自動完成系統將建議相應的文字式程式指令，可以透過「 Enter」 鍵或「Tab」 鍵來選擇。這個系統還會提示學習者輸入函式中參數的正確順序。此外，學習者還可以透過點擊位於右上角的 Python 指令查看器圖標（圖 3d）來查看完整的底層 Python 程式碼。VEXcode 中的每個積木都有一份詳細的幫助檔案，可以透過在積木上方右側的圖標或右鍵點擊積木並選擇“積木幫助”進行訪問，這份檔案解釋了每個積木的功能（圖 3e）並包括了一節關於 Python 指令以參數的詳細資訊（圖 3f）。

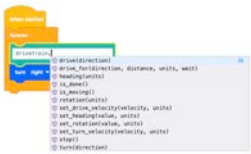
轉換功能旨在幫助程式設計初學者將底層 Python 程式碼與視覺化程式積木進行連結。儘管 VEXcode 提供了一個可顯示完整程式碼的綜合程式碼查看器（圖 3d），但對於那些試圖將 Python 程式碼與相應的視覺化程式進行連結

的初學者來說，可能顯得太過困難。為解決這個困難，Switch Mode 中的每個積木都是可轉換的。學習者可以透過右鍵點擊任意一個積木並選擇“將視覺化程式積木轉換為 Switch Mode 積木”，以查看 Switch Mode 積木相應的 Python 程式碼。這個功能不僅提供了底層 Python 程式碼，而且可以在一個近似於文字式程式語言環境中修改和測試程式碼。此外，如圖 2b 中展示的 Switch Mode 積木是一個可進行堆疊的多行積木，這意味著學習者可以透過「Enter」 鍵或「Return」 鍵在程式碼之間進行跳轉。多行的 Switch Mode 積木是允許進行擴展的，可以參考圖 4c 展示的方式使用。

（a）單行轉換



（b）C 形積木轉換



（c）輸入時自動完成

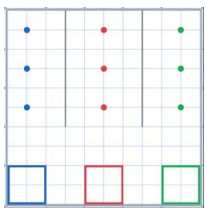
（d）文字式程式的指令查看器

（e）輔助檔案查看器

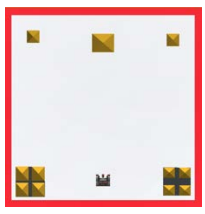
（f）Switch Mode 幫助檔案

圖 3：Switch mode 內置的各種功能

本研究的 8 位學習者參與了一個學期的高中資訊工程程式設計課程，課程採用項目式學習方式，課程目標為學習資訊工程的基礎概念。8 位學習者透過課程作業以及課程項目進行自主學習，學習過程中學習者可以彼此或向教師尋求協助。學習者們使用 Switch Mode 功能完成了四項不同的活動。“磁碟移動”讓學習者控制虛擬機器人在地圖上的預定位置拾起和放置彩色磁碟（圖 4a）。“摧毀城堡”要求學習者編寫程式指令控制他們的機器人推倒一座虛擬城堡（圖 4b）。“導航迷宮”讓學習者導航離開預先設置的迷宮（圖 4c）。最後一項活動是“珊瑚礁淨化”，這是一項一分鐘的任務，挑戰學習者控制他們的機器人在虛擬海底收集盡可能多的垃圾（圖 4d）。結束第一輪僅使用視覺化程式語言環境的五週教學後，學習者們被邀請參與第一次半結構式訪談，詢問他們使用 VEXcode 的經驗。結束第二輪使用 Switch Mode 程式設計環境（圖 1）的五週教學後，學習者們再次被邀請參與第二次半結構式訪談，了解他們在視覺化程式語言和 Switch Mode 中的程式語言經驗。其中一個訪談問題是：“你從視覺化程式語言環境過渡到 Switch Mode 使用什麼程式設計策略？”透過這個問題探討他們在混合程式語言環境中的程式設計策略。

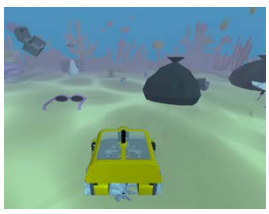


（a）磁碟移動



（b）摧毀城堡

（c）導航迷宮



（d）珊瑚礁淨化

圖 4：不同的任務場地

研究開始前，每位學習者皆完成了一份前測問卷及一份先備知識測驗。前測問卷包含對程式語言自信程度、學習動機、學習參與程度和真實性四個向

度，每個向度各五題；先備知識測驗內容為為關於程式設計的 20 個選擇題。這些數據搜集目的是了解學習者對程式設計的先備知識以及態度。

本研究重點關注兩位參與者，Madison 和 Richard，因為他們代表了兩種不同的背景及過往程式設計學習經驗的學習者。Madison 和 Richard 均為九年級的學生， Madison 已經完成了 HTML、JavaScript 和 CSS 課程，而 Richard 是在中學參加了機器人社團，並擁有包括眾多視覺化和文字式程式語言在內的廣泛程式設計經驗。Richard 還擁有使用 C++ 集成第三方傳感器的豐富 Arduino 程式設計經驗。

肆、研究結果

一、Madison

Madison 是具有部分程式設計經驗的學習者，曾經學習過 HTML、CSS 和 JavaScript 的課程，對於 Scratch 也有一定的熟悉度。Madison 將程式設計定義為「編寫程式碼供電腦引擎讀取和執行。例如，編寫一個網頁程式會涉及使用 HTML 和 CSS 程式碼將內容放到網站上。」儘管有這樣的學習經驗背景，Madison 在訪談中表示對編寫程式碼感到不舒適。關於他們對程式設計的態度，隨時間推移的調查反映出一個一致但不斷演變的觀點。在前測問卷中，Madison 的程式語言自信程度平均為 2.2 （滿分 5 分），學習動機為 2.4 （滿分 5 分），學習參與程度為 3.4（滿分 5 分），真實性為 3.4 （滿分 5 分）。在先備知識基礎評估中，Madison 的前測得分為 15 分（滿分 20 分）。

在訪談中，Madison 分享到在文字式程式語言環境中感到不自在，更喜歡在視覺化程式語言中組合程式積木「我不太擅長，比如 Python 和類似的東西，但喜歡使用視覺化程式積木，能夠像…像…結合在一起，這樣做對我有幫助。」鑑於這種對文字式程式語言的不適感，他在程式設計時經常使用 Switch Mode。特別是對於像 if/else 這樣的條件語句，他僅使用視覺化程式語言模式來編寫程式碼。

> 「我仍然需要…像…查看其他視覺化程式積木並轉換它[將其他視覺化程式積木轉換為 Switch Mode 積木]以確定[我在 Switch Mode 積木中輸入的是正確的命令]……通常我需要轉換的是像 repeat 視覺化程式積木（for loop）或 wait untill...，[我通常輸入]像是 drive 和 turn 的指令[在 Switch Mode 積木中的指令]。所以這些相當簡單....我通常只用這些條件語句程式積木[例如 if/else]。我還沒有嘗試過將 Switch Mode[用於條件積木]。」

相較於其他指令，如 wait until 或 for loop，Madison 有一個特定的方式。他將指令的視覺化程式積木拖到工作區的一側，然後將其轉換為 Switch Mode。然後，他將一個空的 Switch Mode 積木拖到主程式的適當位置並開始在新轉換的積木中輸入指令。

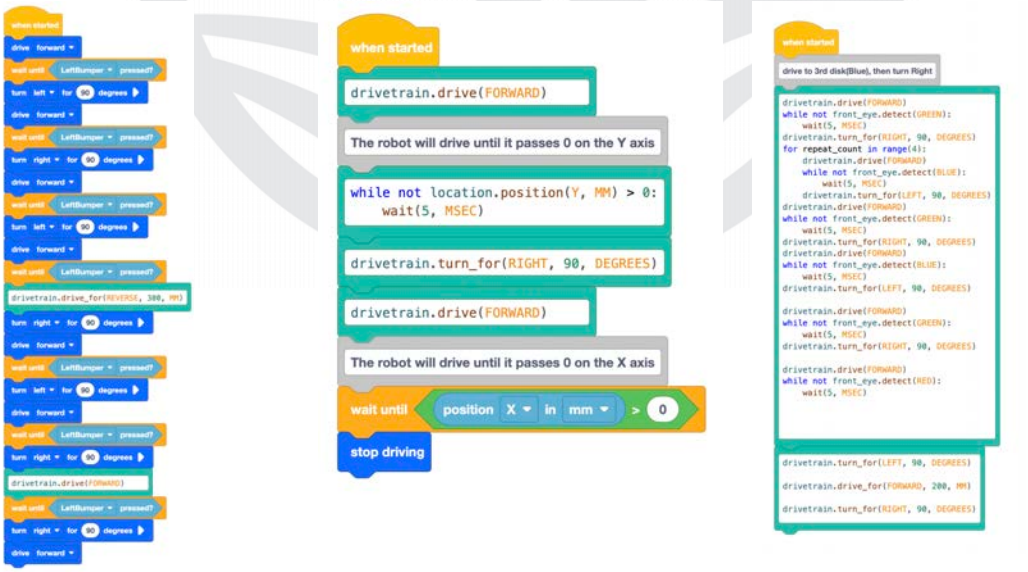
> Madison：我不太擅長編寫指令，我更習慣於從另一個被轉換的視覺化積木的情況中查看它。
>
> 訪談者：好的。所以你的意思是，當你編寫指令時？
>
> Madison：是的。當你在編寫指令的時候。
>
> 訪談者：所以你基本上是拿另一個視覺化程式積木並轉換它。
>
> Madison：是的。然後我看看它長什麼樣子然後輸入它。

Madison 這樣解釋說：「[我]使用[轉換指令]，而不是僅僅在腦海中閱讀程式碼。」他進一步反饋說，在程式碼查看器中查看完整的 Python 程式碼

（圖 3d）令人不知所措，他表示「完整的 Python 程式碼[查看器]，有太多程式碼了。」這也是預料之中的，因為缺乏文字式程式語言經驗的學習者將很難在完整的程式碼查看器內找到相應的 Python 程式碼行數。因此，轉換指令更適合 Madison 的需求，將一半視覺化程式積木和與 Switch Mode 積木內的 Python 程式碼之間進行連結。

Madison 在輸入語法較少複雜的指令時感到更加舒適，例如 `drive` 和 `turn`。他解釋說，「實際上要輸入所有的東西，而不是能夠只輸入如`drive`，`turn`。...這些指令[`drive` 或 `turn`]只有一個，你必須拖出來並輸入你想要的任何東西。它[可能]超過 50 條[指令長]，你知道嗎？」Madison 試圖說明，當拖動 `drive` 或 `turn` 視覺化程式積木時，他們仍然需要修改積木的參數，如距離或角度。對於更複雜的程式碼，可能需要超過 50 次的 `drive` 或 `turn` 程式積木才能完成一項任務（圖 5a）。因此，他更喜歡在 Switch Mode 中輸入它們並在編寫指令時輸入參數。

當被要求進一步解釋為什麼他偏好使用轉換指令時，Madison 說，「如果只有一個積木，你做了轉換。那有幫助。就像它幫助我更好地聯想。」他指出，一次只轉換一個積木使他能夠更容易和直接地將一個指令與相應的 Python 程式碼連結起來。Madison 也表達了他對 Switch Mode 的喜愛。他認為 Switch Mode 是視覺化程式語言和文字式程式語言之間的有效過渡工具，他表示「我實際上認為 Switch Mode 真的很酷，因為它幫助[你]容易地進入實際的程式碼部分。」Madison 也強調了能夠從視覺化程式語言轉換到 Switch Mode 的教育益處。他解釋說，「有時你看到一段指令你必須使用，像一個程式試圖告訴你它做什麼，但就像我仍然不明白它做什麼。所以能夠像轉換積木看看，那做了什麼或看看[底層 Python 程式碼]。我認為學習如何做事情真的很有幫助。」總之，Madison 將轉換功能視為他學習程式設計的鷹架，以及一種讓學習者更容易進入文字式程式語言的方式。



（a）Madison 的長程式　　（b）Madison 的程式　　（c）Richard 的程式

圖 5：Madison 和 Richard 的程式

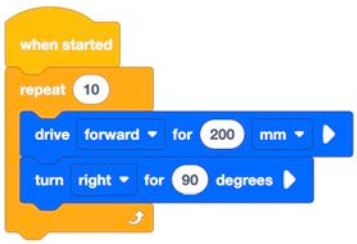
二、Richard

Richard 是擁有 Scratch 到 C++ 等廣泛的程式設計經驗背景的學習者，將程式設計視為一種指令和行動的媒介，並概括這一觀點表示：“對我來說，

程式設計就是給出指令來驅動某事做某事。它是使用某種語言中的精確指令來完成目標。”Richard 的先前程式設計經驗大量涉及閱讀程式碼庫和幫助檔案，如 Arduino 的第三方庫。在他的第一次訪談中，他說：“我必須找到每一個[Arduino 的第三方庫]的程式碼庫。”而在他的第二次訪談中，他說：“我必須學習程式碼庫。”他的程式設計學習歷程始於中學的機器人課程使用 VEXcode，自那以後他的學習動機不斷轉變，具有明確的目標：“今年年初時，我想做的一件事就是學習 Python。”在前測問卷中，Richard 的程式設計自信程度平均為 3 （滿分 5 分），學習動機為 3.6（滿分 5 分），學習參與度為 3.2 （滿分 5 分），真實性為 3.2 （滿分 5 分）。在知識基礎評估中，Richard 的前測得分為 17 分（滿分 20 分）。

儘管 Richard 擁有豐富的程式設計背景，但過渡到 Switch Mode 並非沒有困難。他發現從視覺化程式語言到文字式程式語言的轉換需要一段調整期，坦白來說，“因為如果我直接轉到[Switch]，那我[必須]預測它的結構。我無法理解視覺化程式積木的結構。在我看來，它[積木形狀]太令人困惑。”這反映出當一個擁有豐富基於文字式程式設計經驗的學習者面臨視覺化程式設計結構的意外挑戰時，他的學習路徑將如何呈現。Richard 繼續詳細說明了視覺化程式設計結構，以 loop 為例，“我認為對我來說最大不同之處是可以輸入指令。我認為在視覺化程式積木內部，不能將一個視覺化程式積木放進去後，輕易地把它從 loop 內部切換到 loop 外部，必須拖出原本的視覺化程式積木。但用 Switch Mode，可以只用「backspace」鍵就把它切換出來。”例如，一個包含重複 10 次的程式可以在 loop 內部有兩個指令（圖 6a）。當學習者拖動另一個視覺化程式積木進入 loop，比如 `print ABC` 並 `set cursor to next row`（圖 6b），但後來想把 `set cursor to next row` 移動到 loop 外部，學習者需要點擊該視覺化程式積木，將它拖出 loop，然後放在重複 loop 下方（圖 6c）。如果 loop 處於 Switch Mode（圖 6d），學習者可以只用「backspace」鍵來移除縮進，將它移動到循環外部（圖 6e）。

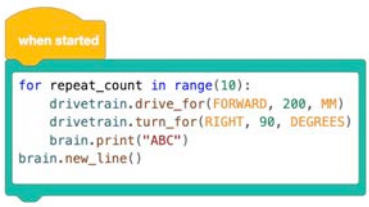
這種理解方式強調擁有不同先前經驗的學習者可能希望從中獲得的不同功能。這一點也在他的第二次訪談中被提及，關於縮進（“spacing”），“通常當你使用一種[文字式程式]語言時，有時你必須放置空格來顯示什麼在[loop]內部以及什麼不會造成混淆。你必須使用註解或類似的方式。如果間隔不正確，它可能成為另一個指令。”



（a）帶有 loop 的簡單積木程式碼

（b）在 loop 內添加兩個額外指令

（c）將最後一個指令移出 loop



（d）多行 Switch Mode　　（e）在 Switch Mode 中使用退格鍵刪除最後一行 Python 程式碼之後

圖 5：視覺化程式積木模式與 Switch Mode 之間的比較

當他對某個視覺化程式積木感到不確定時，他會大量依賴可用的資源，利用幫助檔案作為兩種模式之間的橋樑。“我使用的策略是使用問號[幫助檔案按鈕]來看視覺化程式積木實際意味著什麼。當我使用 Switch Mode[轉換]過來時，我就理解了它的含義，並且可以進行相對應的修改....我猜我在[轉換和輸入]兩者都可以做一些。我轉換它直到我足夠了解該視覺化程式積木。然後我就可以直接輸入了。然後我將它與[幫助檔案]進行比較。那些幫助檔案對我的幫助很大。”這種方法強調了一種平衡的方法論；將積木轉換為文本，直到熟悉使他能夠“直接輸入”，並不斷將他的程式碼與幫助檔案進行比較以加強他的理解。

在他的一次程式設計課程中，他輸入了 `drivetrain.turn(RIGHT, 90, DEGREES)`。然而，`drivetrain.turn` 指令只接受一個參數，而 `drivetrain.turn_for` 指令接受三個參數，正如 Richard 所輸入的那樣。他一定是把這兩個指令搞混了。但這不是他程序中唯一的指令。他花了一些時間修改他的程式，將其與幫助檔案進行比較，並進行了四次試錯，但當時未能找到問題所在。然後他決定在側邊拖動一個向右轉 90 度的程式積木（圖 7a），然後右鍵點擊轉換。接著，他將轉換後的積木 `drivetrain.turn_for(RIGHT, 90, DEGREES)` 與他輸入的積木進行比較，並透過輸入指令修復了問題（圖 7b）。這只是一個小範例，展示了 Richard 如何利用幫助檔案和轉換功能來幫助他編寫程式碼。在課程的中 Switch Mode 部分的後期，他為其中一個活動的程式在頂部有一個單行註解積木，描述程式碼應該做什麼，然後有兩個多行 Switch Mode 積木包含了所有程式碼（圖 5c）。這些顯示了他從輸入指令中取得了更多進步，並且使用 Python 輸入時感到更加自在。

（a）在側面拖動了一個 `turn for` 積木　　（b）轉換了 `turn for` 積木

圖 7： Richard 的程式碼

在我們的訪談中，Richard 的反思為擁有一些文字式程式設計經驗的學習者展示了一種逐步使用 Switch Mode 的操作方式。他擁有豐富的文字式程式語言經驗，在首次接觸視覺化程式語言結構時與視覺化程式的編寫程式碼有些掙扎。在 Switch Mode 課程部分的尾聲，他能夠利用先前透過閱讀所有程式碼庫或幫助檔案的經驗，進行轉換和輸入指令的操作。他的訪談結果展示了一個循序漸進學習過程，證實了正確工具在一個積極的學習者手中的力量。他還分享了學習 Python 過程中充滿成就感的時刻：“我對[Python]理解更深了。”

伍、討論

一、個案分析

從研究開始前的學習態度問卷調查中可以了解到 Madison 的程式設計自信程度和學習動機低於 Richard，他的先備知識基礎評估也低於 Richard。本

研究發現儘管他們有著不同的先前程式設計學習經驗，但兩人都在 Switch Mode 中找到適合他們編寫程式的方式。

Madison 過往擁有基礎程式設計學習經驗，他在課程的開始階段就開啟了強調舒適學習程式語言的學習歷程。Madison 透過在視覺化程式語言學習過程逐漸進步中建立信心，逐漸透過轉換功能將指令轉換到 Switch Mode 中，鞏固對於視覺化程式語言的理解並逐漸過渡到在 Switch Mode 積木內輸入 Python 程式碼的學習。對於他不熟悉或從未使用過的任何指令，他會透過僅使用視覺化程式積木方式來維持舒適的學習程式設計過程中。Madison 案例中，多種模式的結合是混合程式語言環境的關鍵，特別是在 Switch Mode 的環境中格外顯著。

相反，Richard 帶著較完備的文字式程式語言基礎知識進入高中。Richard 的程式設計策略依賴於他與 Arduino 的經驗，他使用幫助檔案來解讀每個積木的底層程式碼。他的學習目標十分清楚——“過渡到 Python”，高學習動機影響他從第一次程式設計課程起對程式設計環境的編寫方式。因此，他使用幫助檔案來理解底層 Python 程式碼，使用轉換功能來查看並修改 Switch Mode 內的 Python 程式碼。一旦他熟悉了指令，他就會直接輸入指令並與幫助檔案進行對照，再次檢查錯誤。

探討 Madison 和 Richard 學習路徑的目的是強調需要滿足對資訊工程教育採取細緻化方法的需求，正如（Weintrop et al., 2020）所提出的。兩位學習者在該學科中因為截然不同的背景和目標導致了在相同程式設計環境中以不同方式編寫程式碼。有鑑於學習者的先前學習經驗和學習動機存在顯著差異，能夠根據學習者當前的水平來定制適合的學習經歷並支持他們持續學習，在資訊工程教育中尤其關鍵。Madison 透過過使用視覺化程式積木逐步建立學習信心，以及 Richard 對文字式程式語言的問訊式方法，展示了先前程式設計經驗如何塑造學習者在混合程式語言環境中編寫程式碼的兩種方式。

二、建議

本研究為設計支持完全視覺化程式語言和完全文字式程式語言的創作工具之間的學習者創作的混合編程環境提供了一種嶄新的方式。從 Madison 和 Richard 的學習歷程中整理出一些研究建議，本研究認為混合程式語言環境具有滿足每個學習者需求的潛力。傳統的程式設計環境通常是在視覺化程式語言和文字式程式語言的兩種模式之間提供其中一種模式，難以提供一個全方位從先前程式設計經驗過渡的鷹架。 Switch Mode 引入了靈活性以容納多種程式設計策略的混合環境，讓學習者能夠在其中選擇他們最適合的編寫方法。

混合型環境的目標是支持學習者從視覺化程式語言過渡到文字式程式語言。本研究雖然側重於兩個個案研究，但目的是展示混合型環境如何支持多樣化的程式設計方式並協調課堂的多樣化教育需求及偏好，讓每位學習者能夠按照學習者能力和目標取得進步。學習關鍵在於允許每位學習者以符合他們先前學習經驗的方式沿著自己的學習路徑取得進步。此方式強調教育工具設計者在創建程式設計環境時考慮廣泛學習經驗範圍的重要性。根據上述兩位個案的經驗，本研究可以確保混合型環境不僅具有包容性，而且更有效滿足學習者不同學習目標。

陸、結論

Madison 和 Richard 的個案研究展示了學習知覺和目標如何與學習者操作程式設計工具的方式相互作用。Madison 從對程式設計感到不舒服到對輸入

一些文字式程式指令產生自信，以及 Richard 利用幫助檔案掌握新的程式語言策略，表明學習者與工具互動的方式與他們的個人學習目標及先前經驗深深交織在一起。

在分析學習者學習知覺和學習目標如何影響工具使用方式的過程中，本研究明確指出教育科技的設計必須緊密關注這種不斷變化的相互影響關係，以確保教育工具能夠有效地滿足學習者需求。Madison 透過視覺化程式語言環境的支持而逐漸建立的自信，以及 Richard 結構性使用幫助檔案來理解和熟悉 Switch Mode，反應出學習者的先前經驗和學習目標決定出的兩條不同學習路徑。

本研究強調設計的教育工具時，工具需要分辨並回應學習知覺和學習目標差異的重要性。這些工具的設計不僅應該促進不同學習階段間的順利過渡，還應該提供一個平台，讓學習者可以適應自己專屬的學習路徑，提升他們在資訊工程領域的學習體驗。

參考文獻

Blanchard, J., Gardner-McCune, C., & Anthony, L. (2020). Dual-Modality Instruction and Learning: A Case Study in CS1. *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, 818–824.

Horn, M. S., Solovey, E. T., Crouser, R. J., & Jacob, R. J. K. (2009). Comparing the use of tangible and graphical programming languages for informal science education. *Proceedings of the 27th International Conference on Human Factors in Computing Systems - CHI 09*, 975.

Kölling, M., Brown, N. C. C., & Altadmri, A. (2017). Frame-Based Editing. *Journal of Visual Languages and Sentient Systems*, *3*, 40–67.

Lin, Y., & Weintrop, D. (2021). The landscape of Block-based programming: Characteristics of block-based environments and how they support the transition to text-based programming. *Journal of Computer Languages*, 101075.

Lin, Y., Weintrop, D., & Mckenna, J. (2023). Switch Mode: Building a middle ground between Block-based and Text-based programming. *Proceedings of the 2023 Symposium on Learning, Design and Technology*, 114–118.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, *10*(4), 1–15.

Moors, L., Luxton-Reilly, A., & Denny, P. (2018). Transitioning from Block-Based to Text-Based Programming Languages. *2018 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, 57–64.

Weintrop, D. (2019). Block-based programming in computer science education. *Communications of the ACM*, *62*(8), 22–25.

Weintrop, D., & Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education*, *18*(1), 1–25.

Weintrop, D., & Wilensky, U. (2018). How block-based, text-based, and hybrid block/text modalities shape novice programming practices. *International Journal of Child-Computer Interaction*, *17*, 83–92.