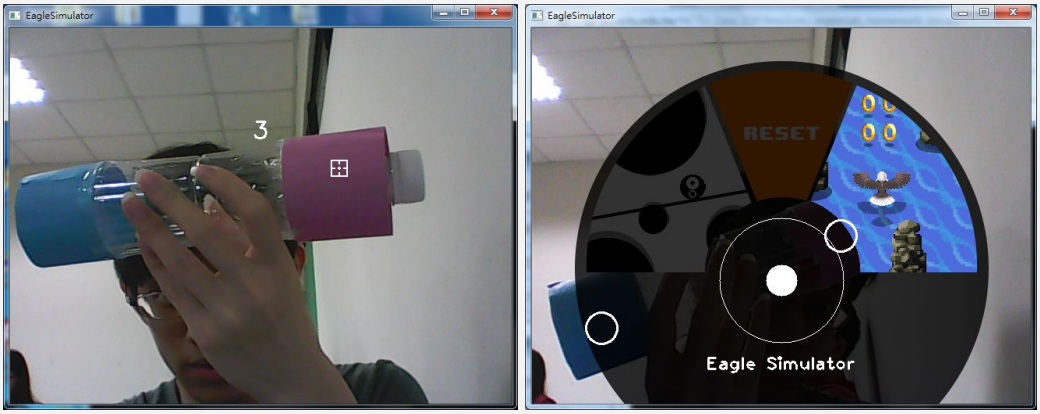
HCI Personal Project Report

0316002 俞達

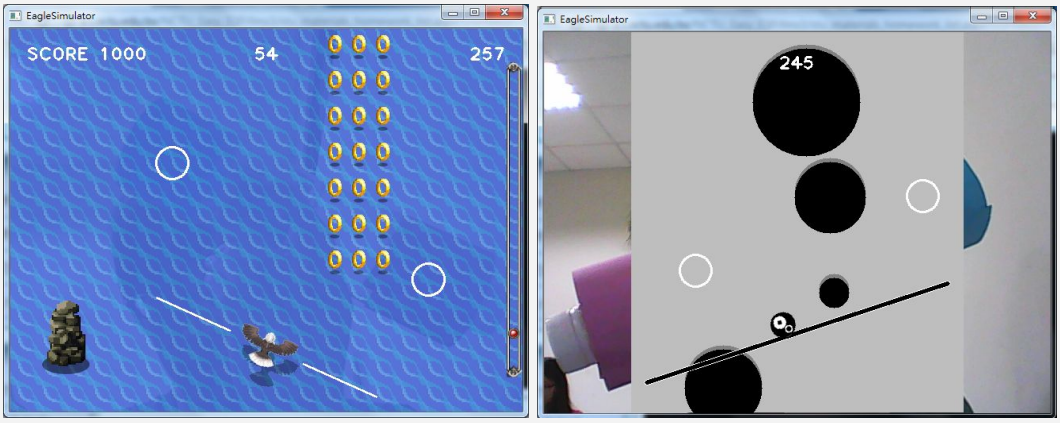
環境：Visual Studio 2013 + OpenCV 2.4.13 + rapidxml

1. Introduction

專案內包含兩個小遊戲：老鷹模擬器和 Rolling Rocker。在老鷹模擬器中，玩家透過旋轉控制器調整老鷹的飛行方向，沿著飛行方向上下甩動控制器讓老鷹加速，並嘗試獲取金幣和避免撞到岩石。在 Rolling Rocker 中，玩家透過旋轉控制器調整平衡桿，Rock 會在平衡桿上滾動，透過滾動 Rock 來避免掉落到地圖上的黑洞中。



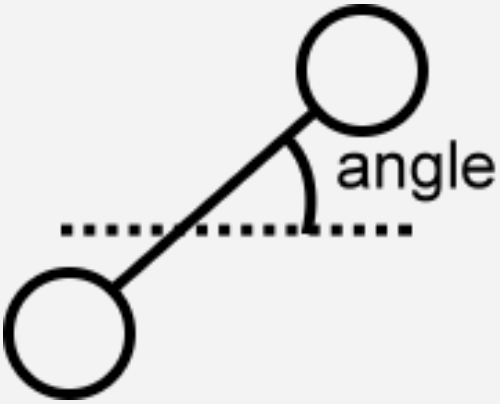
開啟程式時，會先要求玩家偵測控制器的端點顏色(上左)在進入標題場景(上右)。在標題場景中，玩家可以看到選取的兩個顏色所抓取的指標以白色空心圓形顯示。透過改變兩個圓形的相對位置來改變向量的角度。不同的角度會選擇不同的按鈕，停留在按鈕上一段時間後就會執行按鈕的事件。這裡我們可以選擇重新選擇顏色、老鷹模擬器或是 Rolling Rock。

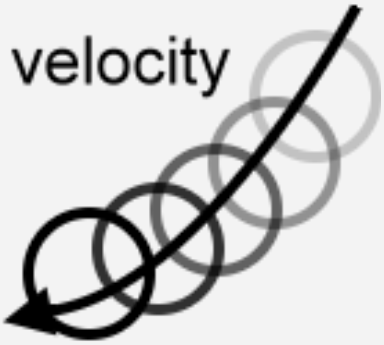


玩家可以在 short_sea.txt 中修改參數來改變老鷹模擬器的格子地圖內容或是地圖總長度。而玩家也可以在 settings.xml 中修改 Seed 參數來改變隨機生成的序列。上左為老鷹模擬器，上右為 Rolling Rocker 。

2. System

Control





透過組合兩個簡單的色彩追蹤物件，產生一個新的控制維度 – 旋轉空間。另外也利用 deltaTime of frames 和 position difference 計算出 velocity 並進一步轉成 motion。

Game Play

老鷹模擬器這一類的遊戲如果用一般的鍵盤滑鼠控制遊玩是蠻常見的。搭配特殊控制器並嘗試組合出新的玩法。起始構想是要玩家在手上綁著翅膀並實際去揮動手臂和旋轉肢體才可以移動。但是因為空間限制，就把原本要大空間遊玩的硬體裝置改成小型且好控制的棍狀裝置。事實上原本構想方法是可行的 (就是把棍狀裝置拆成兩個)，但實際測試時發現單靠 RGB 影像 (或是可能鏡頭不夠好？) 去偵測 Motion 很不準確，過度大幅度的移動會導致影像生成時有殘影而無法正確判斷追蹤目標的位置。

Rolling Rock 是款非常獨特的遊戲，具備新穎的遊玩方法和機制。實作的時候礙於時間和精力，僅把基本的單一物件隨機生成和角色控制給完成，但其實還想再加入更多類型的物件，不單單是讓玩家死亡的物件。由於遊戲中不需要偵測 Motion ，而且大部分的控制的是微量的調整，所以控制會比老鷹模擬器還要來得穩定。

Difficulty

平常在設計製作遊戲時，都是習慣使用 Unity 當作引擎。但這次由於作業限制，但又想在和 Unity 類似的環境下進行實作，所以就挑戰使用自製的遊戲引擎來實作，遊戲引擎的詳細實作會在之後提到。製作簡單的遊戲引擎比主要遊戲內容花上更多時間，但完成引擎的基本功能後，剩下和遊戲相關的實作就進行的比較順利了。除此之外所遇到的挑戰 (Motion Tracking、記憶體用量優化)，都有在上下文中提到和說明解決方法。

3. Implementation

下面會提到我認為比較特別的實作內容，當然除此之外還有些零碎簡單的實作。

DY Game Engine

雖然說是遊戲引擎，但沒有 GUI 編輯介面，主要提供一個 Component-based 的物件管理環境和功能。GameEngine 最上層的物件是 SceneManager，負責管理 Scene，而每個 Scene 底下都有一個 ObjectManager，ObjectManager 底下有該場景的所有遊戲物件 (Object)，每個遊戲物件中都可以擁有多個組件 (Component)。每一個 MainLoop 中，會先計算上一個 frame 和這一個 frame 之間差了多少時間 (timeStep or deltaTime)，然後在 call SceneManager 的 Update(timeStep) 並在一層一層叫下去，Update 中主要負責 GameLogic 的更新。Update 結束後再 call LateUpdate(timeStep)，LateUpdate 主要負責和 Rendering 相關的 Logic。利用這套系統，使用者可以輕易切換讀取不同的 Scene，因為 Engine 內部已經負責好 Memory Management，確保不會發生記憶體不足的狀況。我也同時寫了二維向量運算的 library 以方便 Logic 的運算和實作，並利用 OpenCV 實作簡單的 Graphics System，功能包含 SpriteRendering, ColorMask, Alpha Blending 等等。

Color Tracking

控制器是利用色彩區塊來追蹤 (詳細程式碼見 class BallTrackingComponent 和 class ColorSelector)。首先，取得了 BGR 空間影像畫面後，轉換成 HSV 空間。將在一個 HSV 範圍內(由 ColorSelector 取得) 的 Pixel 都設成 True，其餘 False。得到 Mask 之後進行 Erosion 和 Dilation 讓實際上可能相連的但因為 Noise 而分開的 connected-component 連起來。之後取得 components 的輪廓，並取輪廓最大的 connected-component 計算 Minimum Enclosing Circle 的半徑和中點。若半徑符合一個 threshold，就將此 component 視為目標物。

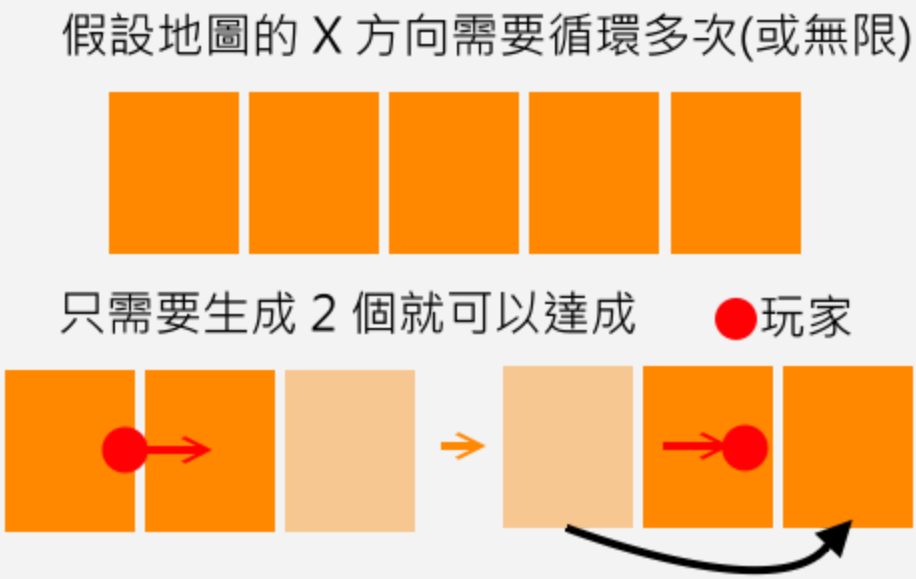
Frame Rate Issue

一開始在製作 Game Engine 的時候，我將 Video Capture 的 Logic 一起放入 Engine 的 Main Loop 中，導致 FPS 不到 20 甚至下降到 10 以下的狀況。主要的 Overhead 在於 OpenCV 的 Video Capture 很花時間。為了解決這個問題，我將 Video Capture 的工作放入一個新的 Thread。只有在 Video Capture 產生新的影像的時候，才將 Game Engine 的 Target Frame Mat 換成新的影像，否則就使用舊的影像。這樣可以讓遊戲的 FPS 上升至 60 到 70，而且有新影像時也可以及時使用作 Tracking 和 Rendering 的新目標。

Object Pool

兩個遊戲都是隨機產生相似物件，因此物件的數量管理和記憶體用量有很大的關係。沒有在使用的物件就先 Disable，要求生成新的物件時，就優先從舊的沒有使用的物件來生成。避免在遊戲過程中有多餘的 Memory Allocation 和 Memory Releasing。

Map Division



4. Experiment / Conclusion

在實作 ImageComponent (畫圖片的組件) 的時候，在計算旋轉矩陣時遇到一個很嚴重的問題。首先來講講我是如何實作的，一個 ImageComponent 會存著一個原始 Image 的 Mat。初始化的時候，會依據物件的 Rotation 並利用 原始 Image 產生一個新的 OutputImage。每一個 frame 都會檢查 Rotation 是否有改變，如果有，就要重新計算 OutputImage。但我發現 OpenCv 在使用 remap 和 warpAffine 的時候，會有 Memory Leak (http://code.opencv.org/issues/2502)。而在場景剛產生時，每一個有 Image 的物件都會使用一次 warpAffine。而在老鷹場景中，儘管我已經實作了地圖循環和切割，物件數可能高達上千個 (地圖格子 32 X 32，單一畫面中至少有 (640 / 32) * (480 / 32) = 300 個地圖格子)。導致讀取一次老鷹場景，就會大約 Leak 10^3 KB 的記憶體。最後解決方法算是治標不治本，再生成 Image Component 的時候，可以設定要不要將 Object 的 Rotation Apply 到 Image 上。所以註定不會旋轉的物件 (像是地圖格子) 就設成 False，也就是會直接將原始 Image 當作 OutputImage而不經過 warpAffine 運算。因為大部分的物件數量都來自於地圖格子，因此也算間接了解決 Memory Leak 的問題。