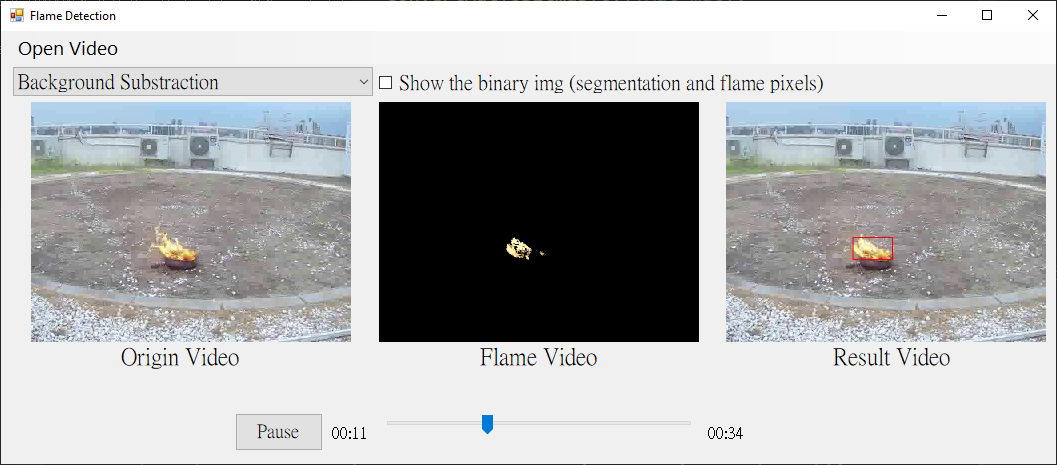
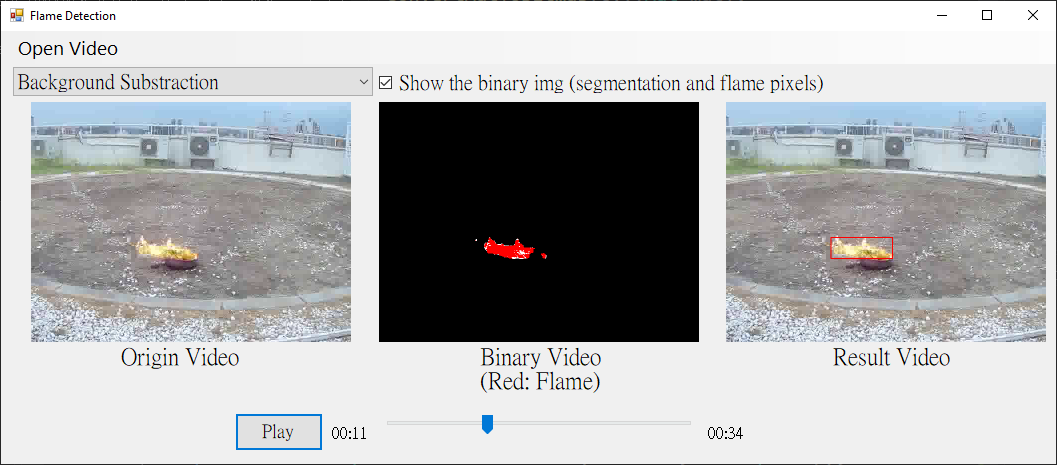
程式執行畫面：

白天：

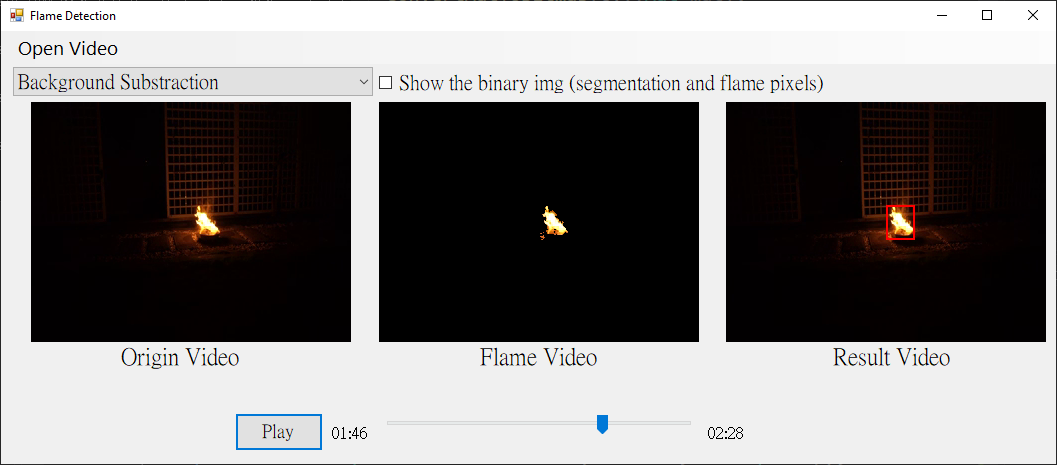
* 擷取火焰



* 顯示二值化影像（白色為移動物，紅色為火焰像素）



夜晚：



（皆可以顯示二值化影像，且也可選擇不同的移動物切割方法）

功能概述：

* 基本功能延伸第二題（移動物切割）：
  + 影片播放、暫停，拉時間軸能夠快轉影片進度。
  + 能夠選擇**「背景相減法（Background Subtraction）」**或**「相鄰相減法（Frame Difference）」**計算，並使用Morphology形態學影像處理來讓二值化影像好處理。
  + 中間的PictureBox，預設是顯示分隔出來的移動物。上方的CheckBox可以切換顯示二值化影像。
  + 右邊的PictureBox是框選結果，我是使用圖學理論中的「**Connected-component labeling」演算法**分隔出各物件。我框選使用了兩種方式：

1. 使用遞迴的**DFS**，因此有些情況可能會Overflow，不過在自家電腦測試時，用提供的兩個測試影片，都是正常的。
2. 使用非遞迴的**BFS**，相比遞迴，較不會發生Overflow了。這邊try了個opencv官方提供的測試影片，原本用遞迴跑這影片一定會Overflow，用非遞迴式來做就正常了，故使用此方法
   * + 程式中有設定常數，僅會圈選面積超過10的目標，來避免圈到不重要的小東西。

* 火焰偵測
  + 先找出移動物，可以選兩種移動物切割方法之一，再判斷火焰像素。
  + 透過條件判斷火焰像素，在二值化影像中用紅色標示火焰像素的位置，以便後續**框選**處理。

程式說明：

* 我將函數宣告與實作拆開，分別寫在.h與.cpp中。
* 與影片相關的資訊（如：fps、幀數、處理過的影像幀、一些常數等），宣告在VideoInfo.h這個檔案內。
* 以下簡單講解主要功能函式：
  + 上一題的程式（移動物切割）

1. System::Void Init();

初始化一些東西，如影片時間軸歸0、取得背景相減法的背景。

1. Bitmap^ BackgroundSubtraction(cv::Mat mat, cv::Mat\* binMat);

背景相減法實作，與相鄰相減法相似，僅相減物不同，因此有下方的GetDiff()函式做整合。

1. Bitmap^ FrameDifference(cv::Mat mat, cv::Mat\* binMat);

相鄰相減法實作，與背景相減法相似，僅相減物不同，因此有下方的GetDiff()函式做整合。

1. Bitmap^ GetDiff(cv::Mat mat, cv::Mat diffTarget, cv::Mat\* binMat);

取得差值，diffTarget是被減的影像，若是背景相減法則是背景，相鄰相減法則是前一幀。兩影像相減，有給誤差空間（程式碼內是設50左右），相減後的像素在此範圍內都視為是移動物，並會設定二值化影像給binMat。

1. System::Void SelectObjects(Bitmap^ binaryBmp);

傳入二值化影像圈選火焰物件，一開始會先跑過Connected-component labeling演算法，會將相鄰的火焰像素（我程式中設定的閥值為大概15個像素內）以同一編號標示，跑完後會得到總火焰物件數，就能取得各火焰物件的邊界，並框選。

1. System::Void ConnectedComponentLabeling\_DFS(BitmapData^ bmpBD, int x, int y, int num);

Connected-component labeling演算法核心，使用**遞迴**的dfs實作，傳入的x y是起點，會不斷找出附近15格內（程式碼內有設Threshold）的像素，並標示編號。

1. int ConnectedComponentLabeling\_BFS(BitmapData^ bmpBD, int num);

Connected-component labeling演算法核心，使用**非遞迴**的bfs實作，會不斷找出附近15格內（程式碼內有設Threshold）的像素，並標示編號。

1. System::Void DrawArea(Bitmap^ binaryBmp, int num);

畫框框的函式，每個物體的框選都會呼叫一次。

* + 火焰偵測

1. System::Void MarkFlamePixels(cv::Mat mat, Bitmap^ binaryBmp);

在二值化影像上用「紅色」**標示火焰像素的位置**，白色則是原本移動物分割的像素。可於畫面中第二格pictureBox中顯示，方便Debug。

1. Bitmap^ GetFlamePixels(Bitmap^ markedBmp, Bitmap^ oriBmp);

取得火焰像素，用以將火焰切割出來並顯示。

1. bool IsFlamePixel(Byte\* bmpPtr);

判斷該像素（bmpPtr）是否為火焰像素。透過以下規則：

Rule 1: R > R\_T

Rule 2: R >= G > B

Rule 3: S >= (255 - R)\* S\_T / R\_T

Rule 4: I > I\_T

並用S

20 <= S <= 100代表較亮環境

30 <= S <= 100代表較暗環境

若是較亮環境： Rule1 && Rule2 && Rule3 == True 即為火焰像素

若是較暗環境：Rule1 && Rule2 && Rule3 || Rule4 == True 即為火焰像素

其餘可看程式內註解，一些細節皆有用註解標示。