COMP 4102- Final Project

Group members:

1. Henok Gebremedhin - 100982033
2. Nazimul Hoque - 100974599
3. Jonathan Lim - 100937925
4. Mohamad-Salim Merhi - 100977984

Table of Contents

Abstract

The project involves 2 main steps and is associated with games. The first step revolves around image recognition of the game “Garry’s Mod” which was replaced from our initial game “Garry’s Mod”. This will look at the health bar over time and convert the number into health. The second step would be to create event handlers that will perform actions to the character at certain health. In a general application, this would be similar to an AI driven by computer vision which could work for multiple games.

Introduction

This is quite challenging converting images into text firstly because none of us has ever worked with text or image recognition. Secondly, we are combining this challenge with game implementation, which only further adds to the difficulty of the process. Text or image recognition is quite a large usage within the computer vision field. Many different variations of such things can be quite useful to a lot of applications in real life. As stated in class, even self driven vehicles require image recognition to look for lines and objects. We will be looking into OCR Tesseract which has some features that can convert images into text. The algorithm we will be using takes images in rounds of the health bar, which is converted to text. We can then check what that value is and apply events when certain numbers are hit. These events will be laid back to the character in the game which will be key presses where the characters action will take place. We will also be using object tracking to track other players or AI opponents. This means that we can trace the opponent if they move for shooting back at them if they shoot us.

Once you are able to convert the image numbers into text numbers, you can do anything you want with it. We purposefully decided to use those numbers for event handling which makes it sort of an AI. This image to text can be used for basically anything involved with vision applications. For example if someone can’t see, you can convert the images into text and turn that text into text-to-speech.

Object tracking is very important as well in vision application. You need to be able to find objects to be able to focus on them and manipulate those objects, because computers only see numbers. Another example of this would be license plate or road signs recognition. If you are able to detect these objects, they can be laid back to image to text and thus the same concept of alternative functionalities.

Background

OCR Tesseract is an open source text recognition engine. It uses an API to extract printed text from images. This open source is based in Python. The general idea of OCR or optical character recognition, takes an image and parses data out of the image by collecting information such as numbers and letters.
https://nanonets.com/blog/ocr-with-tesseract/
This library can be used for different applications like banking, legitimate industry, home and office robotization, etc. Different employments of OCR incorporate identification approval, mechanized number plate acknowledgement and more.
http://www.journalimcms.org/wp-content/uploads/6-Optical-Character-Recognition-with-Tesseract.pdf

OpenCV object detection using HSV color space is used in many applications. HSV color, or Hue, Saturation, and Value, is an alternative representation of RGB color models much more similar to human perspectives. In the link below is a paper on real-time object recognition in robotic applications. HSV color space seems to be the best color space to use in robotic applications because it separates light and chromatic information and distorts the chromatic information the least.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.888.3675&rep=rep1&type=pdf
Here is another paper on moving shadow elimination using HSV color space and uniform extended scale invariant local ternary pattern. The purpose was used for video surveillance, traffic detection, navigation and more which gets rid of shadows to further enhance the actual object.
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9051789

Lastly we use event handlers by simulating keyboard and mouse actions in a listener. This is taken from pynput library. Pynput has a keyboard controller and a mouse controller for press and release of whatever button is on a keyboard.
One issue we had was that we were able to do keyboard input but Garry's Mod didn't process these inputs so we had to search up how to convert simulated inputs to direct inputs. Instead of using the keyboard characters to simulate the click, we need to use hexadecimal keys for the direct input.
https://stackoverflow.com/questions/14489013/simulate-python-keypresses-for-controlling-a-game

Similar ideas of using OCR are 2 videos
https://www.youtube.com/watch?v=FUXNcEF6Pcs&feature=youtu.be&t=
And
https://www.youtube.com/watch?v=D75ZuaSR8nQ
Which are 2 videos with OCR tesseract which detects the health of the game Fortnite, however they convert the values into mechanical outputs such as Alexa and a paintball gun.

Approach

We developed a python script to load and capture a tesseract image, binarize the image and pass it through the Tesseract OCR system (Line 34). OCR Tesseract as stated above is a text recognition engine that converts image into a parsed text data.

The movement of the character is controlled by the cursor which is set to 30 pixels below the crosshair (line 46). The PyAutoGUI is a cross-platform GUI automation used to control the mouse and keyboard (line 47). The x and y coordinates of the cursor was calculated using dx = dx*0.5 and dy = dy*0.5 (line 49-50). The mouse is moved with a mouse button being held down. The current position of the mouse pointer is provided in the x and y members of the event object passed to the callback, i.e. x, and y (line 62). A mouse button is pressed with the mouse pointer at the point selected (line 64).

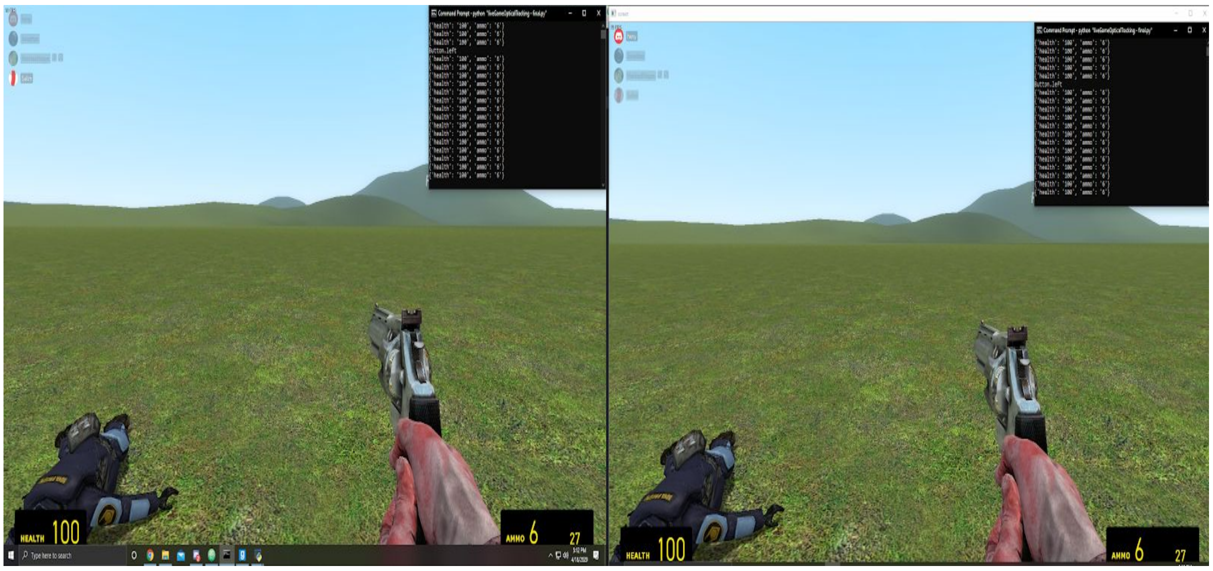
Trackbars were created for tuning the images for contour detection (line 94 – 107). For cv.createTrackbar() function, first argument is the trackbar name, second one is the window name to which it is attached, third argument is the default value, fourth one is the maximum value and fifth one is the callback function which is executed every time trackbar value changes. The callback function always has a default argument which is the trackbar position.

The image is then converted into grayscale by calling the cvtColor function. This function receives the original image and the color space conversion code. We converted the original image from the BGR color space to gray using COLOR_BGR2GRAY.

The hsvContourDetection function which is implemented for image tuning using morphological operations (line 227-228). Morphological operations apply a structuring element to an input image and generate an output image (line 323).

The listener context manager allows you to allocate and release resources precisely when you want to. In this application the context manager encapsulates the original while loop to detect the mouse events. The pyautogui package is used to control and monitor the mouse. The manager can scroll, move, or click on the ammo and health.

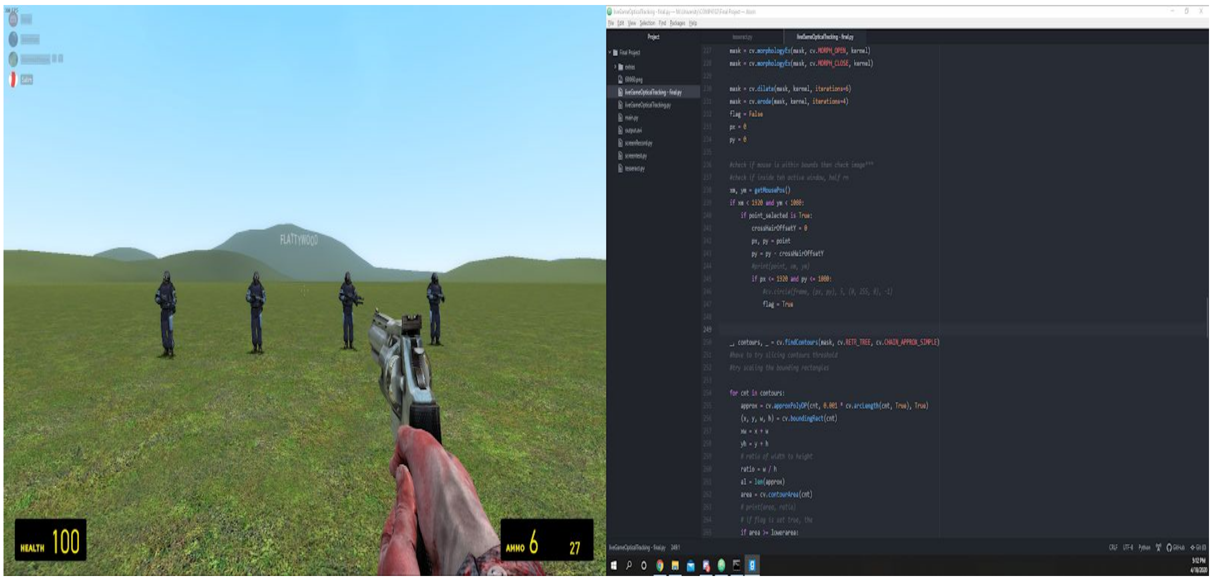
Results:



```
Command Prompt - python "liveGameOpticalTracking - final.py"
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
Button.left
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
```

```
Command Prompt - python "liveGameOpticalTracking - final.py"
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
Button.left
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
{'health': '100', 'ammo': '6'}
```

Figure 1: Screenshot



```
    mask = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
    mask = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)

    mask = cv.dilate(mask, kernel, iterations=6)
    mask = cv.erode(mask, kernel, iterations=4)
    flag = False
    px = 0
    py = 0

    #check if mouse is within bounds then check image***
    #check if inside teh active window, half rn
    xm, ym = getMousePos()
    if xm < 1920 and ym < 1080:
        if point_selected is True:
            crossHairOffsetY = 0
            px, py = point
            py = py - crossHairOffsetY
            #print(point, xm, ym)
            if px <= 1920 and py <= 1080:
                #cv.circle(frame, (px, py), 5, (0, 255, 0), -1)
                flag = True


    _, contours, _ = cv.findContours(mask, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
    #have to try slicing contours threshold
    #try scaling the bounding rectangles

    for cnt in contours:
        approx = cv.approxPolyDP(cnt, 0.001 * cv.arcLength(cnt, True), True)
        (x, y, w, h) = cv.boundingRect(cnt)
        xw = x + w
        yh = y + h
        # ratio of width to height
        ratio = w / h
        al = len(approx)
        area = cv.contourArea(cnt)
        # print(area, ratio)
        # if flag is set true, the
        if area >= lowerarea:
```

Figure 2: Screenshot

List of Work: “equal work was performed by all project members.”

GitHub Page:

https://github.com/henygebremedhin/4102-project

References:

(2013, Jan 5). Trackbar as the Color Palette. Retrieved from https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_gui/py_trackbar/py_trackbar.html

(2018, June 2). Python OpenCV: Converting an image to gray scale. Retrieved from https://techtutorialsx.com/2018/06/02/python-opencv-converting-an-image-to-gray-scale/

(2019, July 28). TAG ARCHIVES: CV2.MORPHOLOGYEX. Retrieved from https://theailearner.com/tag/cv2-morphologyex/

(2019, Feb 23). Handling the mouse. Retrieved from https://pythonhosted.org/pynput/mouse.html

(2019, March 18). Handling the keyboard. Retrieved from https://pythonhosted.org/pynput/keyboard.html

(2019, December 31). Drawing Functions. Retrieved from https://docs.opencv.org/2.4/modules/core/doc/drawing_functions.html

(2019, March 31). Object detection via color-based image segmentation using python. Retrieved from https://towardsdatascience.com/object-detection-via-color-based-image-segmentation-using-python-e9b7c72f0e11