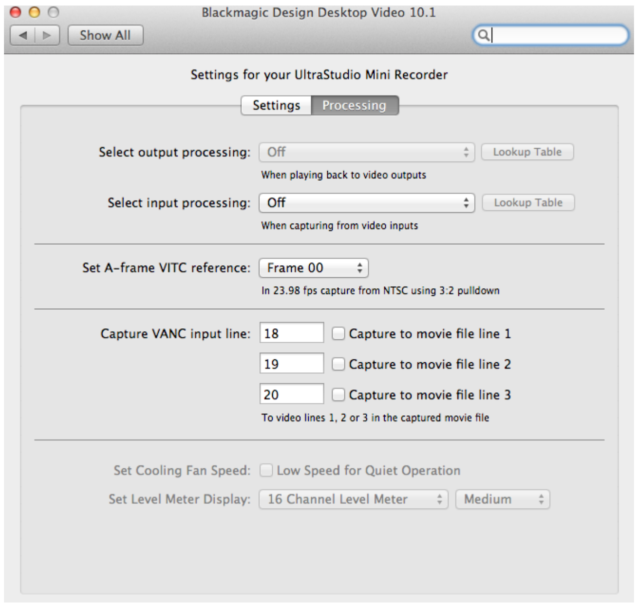
Crowd Participation

By: Tim Lupo

The following will be in depth instructions on how to achieve crowd participation utilizing OpenCV, C++, and Unity 3D. It will include instructions on connecting a HD camera to this program (or using it as a webcam), in addition to in depth code analysis and explanation. It will also include common bugs and how to fix them. Let's get started.

HDMI Camera Connection

To achieve optimal crowd participation, it is very important to have a high resolution camera. Such cameras often have an HDMI output port, where it can stream live video out of. However, Macs do not have HDMI video inputs to process this. The only natively supported external camera video feed is via FireWire, which does not support such high resolution. This video transfer also must be near instantaneous, as every millisecond of lag significantly takes away from the crowd participation spirit. The Thunderbolt port, which is on every Mac after 2011, can both display and receive data (receive is input and input is what we need) and is the Mac's fastest port, so thus is the obvious choice. However, to translate the HDMI input to Thunderbolt, I used an adapter called the BlackMagic Studio Recorder Mini (http://www.amazon.com/Blackmagic-Design-UltraStudio-Mini-Recorder/dp/B009D91314). This is the obvious choice, as it is the perfect mobile external solution that does not compromise the video quality from the HD camera. The Mac Mini already has all of the necessary drivers installed, but if you are using another machine you must install the drivers. To do this, take the included disc and insert it into your machine. Then open its window and go to "Desktop Video Mac" folder. From there you can access the installer, uninstaller, PDFs, etc. There are other installers for various purposes, but you can ignore those. If you lost the disc, or the software is old, you can also go to https://www.blackmagicdesign.com/support to download all software and drivers. To check everything was properly installed, look under apps for the "BlackMagic LiveKey", "Blackmagic Multibridge Utility", "Blackmagic Media Express", and/or "Blackmagic Disk Speed Test." There will also be an option in System Preferences under "Other" (called Blackmagic Design). The next step is to plug in the Blackmagic UltraStudio Mini Recorder. To do this take the input HDMI cable and plug it into the HDMI slot (marked "HDMI IN") on the Blackmagic UltraStudio Mini Recorder. Then take the Thunderbolt cable, and plug one side into the Blackmagic UltraStudio Mini Recorder and the other into your Machine's Thunderbolt port (marked with a little lightening sign). Next turn on your camera, make sure it is on video feed mode, and go back to the computer. There are a few settings that you need to change, as the Blackmagic UltraStudio Mini Recorder also supports SDI input. Simply go to System Preferences > Blackmagic Design (under "Other"), and set the input to HDMI Video & HDMI Audio. You can also tinker around with the other settings, but the default is usually sufficient. Below are images on how it should look.


Blackmagic Design Desktop Video 10.1

Settings for your UltraStudio Mini Recorder

Settings | Processing

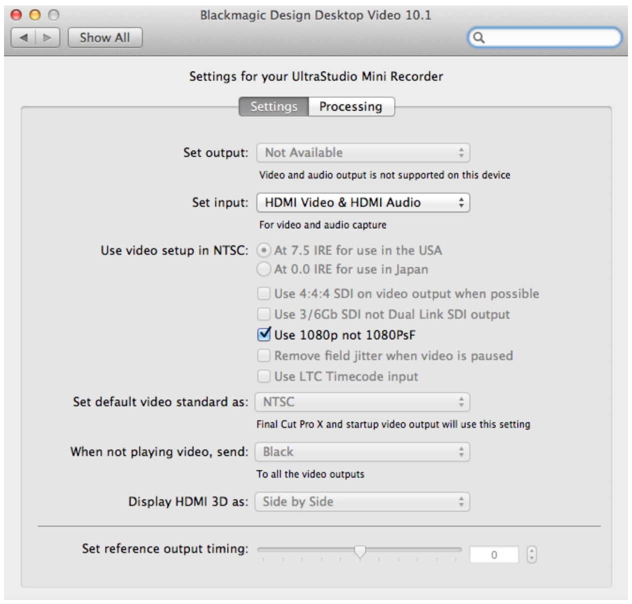
Select output processing: Off — Lookup Table
When playing back to video outputs

Select input processing: Off — Lookup Table
When capturing from video inputs

Set A-frame VITC reference: Frame 00
In 23.98 fps capture from NTSC using 3:2 pulldown

Capture VANC input line: 18 — Capture to movie file line 1
19 — Capture to movie file line 2
20 — Capture to movie file line 3
To video lines 1, 2 or 3 in the captured movie file

Set Cooling Fan Speed: Low Speed for Quiet Operation
Set Level Meter Display: 16 Channel Level Meter — Medium


Blackmagic Design Desktop Video 10.1

Settings for your UltraStudio Mini Recorder

Settings | Processing

Set output: Not Available
Video and audio output is not supported on this device

Set input: HDMI Video & HDMI Audio
For video and audio capture

Use video setup in NTSC: [● At 7.5 IRE for use in the USA] At 0.0 IRE for use in Japan

Use 4:4:4 SDI on video output when possible
Use 3/6Gb SDI not Dual Link SDI output
[✓ Use 1080p not 1080PsF]
Remove field jitter when video is paused
Use LTC Timecode input

Set default video standard as: NTSC
Final Cut Pro X and startup video output will use this setting

When not playing video, send: Black
To all the video outputs

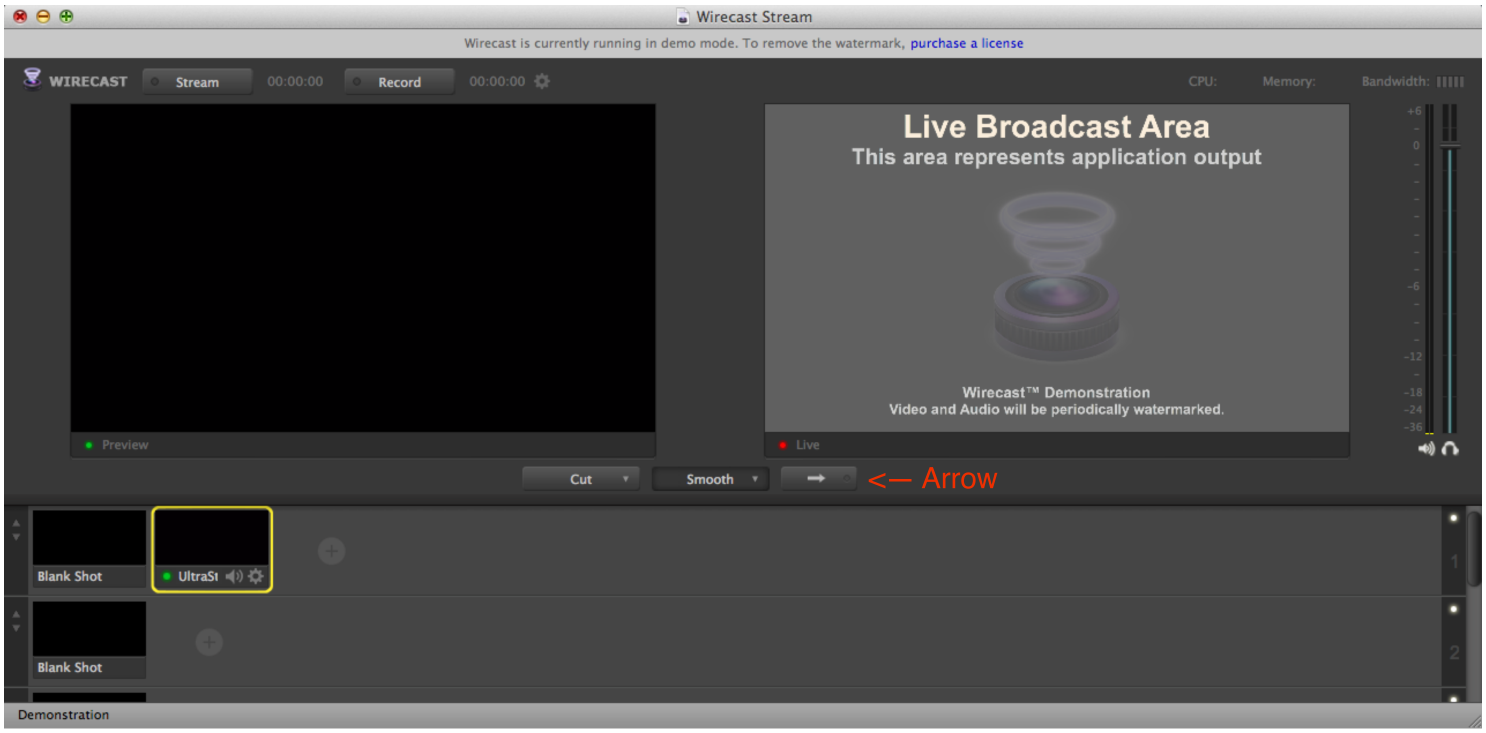
Display HDMI 3D as: Side by Side

Set reference output timing: 0

After this is all done you can test! Launch the Blackmagic Media Express app, and your stream should be in the main window. If it is not there, try switching the tab to “Log and Capture” or back to “Playback.” If it is still not there, make sure all of the cables are plugged in correctly, and the camera is on and in streaming mode (not recording anything). Now we have a live video stream on our Mac, but how can we reference that in our app? The API we will use, called OpenCV, runs off of your webcam. The most straightforward and simple way to use our HDMI video stream would then be to convert it to a webcam! There are other ways around this, like the Blackmagic SDK, but at the time of creation it was under-documented, buggy, and too complicated. To convert our HDMI stream to a webcam we will use a program called Wirecast. There are similar programs, however not for Mac, but Wirecast is a streaming tool that can read capture card inputs (like Blackmagic StudioRecorder Mini) and utilize them to stream. It also has a virtual webcam feature, which allows this streamed video to be utilized as a webcam. To set this up launch Wirecast (or download it here- http://www.telestream.net/wirecast/overview.htm) and create a new project (File > New). There is also a pre-setup file called Wirecast Stream under /Users/admin/Documents/Halloween/Wirecast Stream.wcst, but this is prone to change. After creating a new Project, click the plus button (shown below) and click the video sign. Then choose “Add UltraStudio Mini Recorder Shot.” This will add the stream to your Wirecast Project, then click (it will be highlighted yellow) to move it to the pre-stream. Click the Arrow button so it moves to the right screen. It is now being streamed, but not to the webcam. To port it to the webcam click Output > Virtual Camera Out > Select HD 720p (1280x720) > Start. Now you have set up the webcam and are ready to move to the code! Once you set it up the first time, follow the instructions below to set up the camera and webcam each time.

Steps:
1. Plug in the UltraStudio Mini Recorder
2. Plug the camera into the UltraStudio Mini Recorder
3. Enter Wire cast
4. Go to File > Open Recent > Your File and open it
5. Click the UltraStudio Preview below to highlight it yellow
6. Click the Arrow to move it to the right screen

7. Go to Output > Virtual Camera Out > Start
8. Enjoy and get coding!



Installing OpenCV

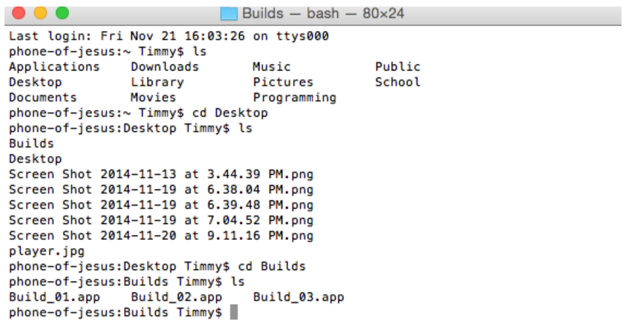
The goal of this project is to create a crowd participation project. To do so, we must analyze the video feed of the crowd for some type of unique characteristic. There are many possibilities, but the cheapest and most efficient is some type of color recognition. There are a couple APIs that assist this, but the best is definitely OpenCV. It has iOS, Android, Java, Python, and C++ support; but since we are using Macs the best way to go is with the C++ build. The following links of how to build the OpenCV library on your Mac. If you are using the Mac Mini version 2.4.9 is already installed, so do not worry about installing it again (unless you need to update).

Blog- http://tilomitra.com/opencv-on-mac-osx/
Video- https://www.youtube.com/watch?v=0aVsZeuemLE

Between the two of these, you should get a good idea of how to install the OpenCV C++ build. However, if these are not efficient, I will condense these into what worked for me-

Steps:
1. Download Xcode command line tools for OSX. It should already be installed, but if it is not make sure to install it.

2. Go to http://opencv.org/downloads.html and download your build. I suggest not choosing Beta builds, but if you need to you can try it. This tutorial will be tailored for Version 2.4.9 for Mac OSX 10.8
3. Go to http://www.macports.org/install.php and download the build for your operating system
4. Install Macports on your computer, following the dmg attached
5. Launch "Terminal" (app pre-installed)
6. You can check to see if Macports successfully installed by typing "port" and hitting enter
7. Now you must install Cmake. To do so type "sudo port install make" into Terminal and hit enter. After it installs, double check to make sure it installed by typing "cmake" and hitting enter
8. Now we must make a directory for the OpenCV build. Move your downloaded OpenCV files into a permanent location (if you move it the library will be messed up)
9. Navigate to your OpenCV files in Terminal. To do this type "ls" to list your folders. Then you must change directories. To do this do "cd ______" (_____ is your folder). Look here ——————————————>



```
Last login: Fri Nov 21 16:03:26 on ttys000
phone-of-jesus:~ Timmy$ ls
Applications    Downloads       Music           Public
Desktop         Library         Pictures        School
Documents       Movies          Programming
phone-of-jesus:~ Timmy$ cd Desktop
phone-of-jesus:Desktop Timmy$ ls
Builds
Desktop
Screen Shot 2014-11-13 at 3.44.39 PM.png
Screen Shot 2014-11-19 at 6.38.04 PM.png
Screen Shot 2014-11-19 at 6.39.48 PM.png
Screen Shot 2014-11-19 at 7.04.52 PM.png
Screen Shot 2014-11-20 at 9.11.16 PM.png
player.jpg
phone-of-jesus:Desktop Timmy$ cd Builds
phone-of-jesus:Builds Timmy$ ls
Build_01.app    Build_02.app    Build_03.app
phone-of-jesus:Builds Timmy$ 
```

10. Once you have navigated to your opencv folder (in my case it is called "opencv-2.4.9"), copy the following command line

```
mkdir build
cd build
cmake -G "Unix Makefiles" ..
```

11. The first line created a directory (folder) named "build", the second line navigates to "build", and the third line sets the foundations for the OpenCV framework in the directory "build"
12. Once you have finished we must install OpenCV. To do this type the following ——————————>

```
make -j8
sudo make install
```

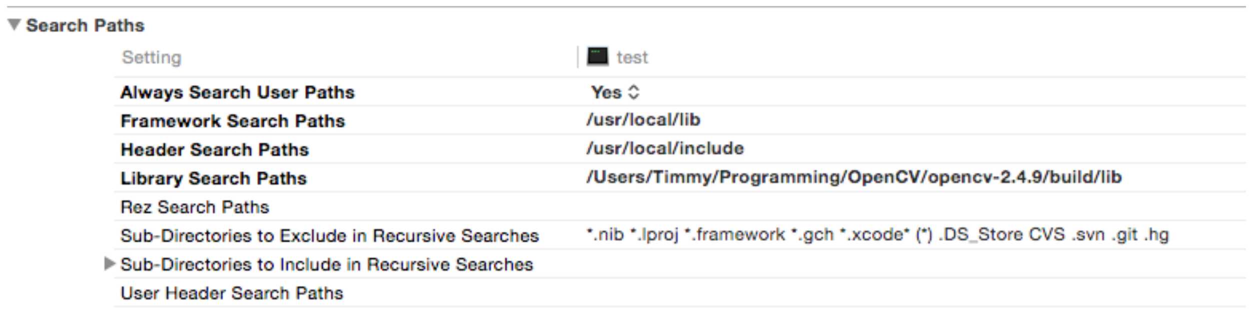
13. Congrats! You have successfully installed OpenCV!

Setting Up an Xcode Project with OpenCV

The last chore before the coding commences is properly setting up a Xcode project to compile OpenCV code. Before we get started, lets discuss why we are using Xcode to compile our code. OpenCV can be compiled in a number of different ways, including terminal manipulation. However, the most organized and efficient way is a large-scale project manager that can handle all of the features we might want to add. For example, utilizing Xcode will allow us not only to build our OpenCV app, but to build a simple interface for others to use later. This is a bit more complicated, and will be discussed as a bonus at the end (it is not necessary and does not effect performance but simply makes the app easier to utilize). The only downside is that Xcode is only compatible with OSX, so if in the future you are using PC I suggest utilizing Microsoft Visual Studio. The following steps will give you an in depth explanation of how to set up a new OpenCV Xcode project, and I will also include a brief video demo-

Steps:
1. Launch Xcode and go to File > New > Project. You will be prompted with a template menu and go to OSX > Application > Command Line Tool
2. Name your project and make sure to select your language as C++, then select your destination for your project (it doesn't matter where you put it)

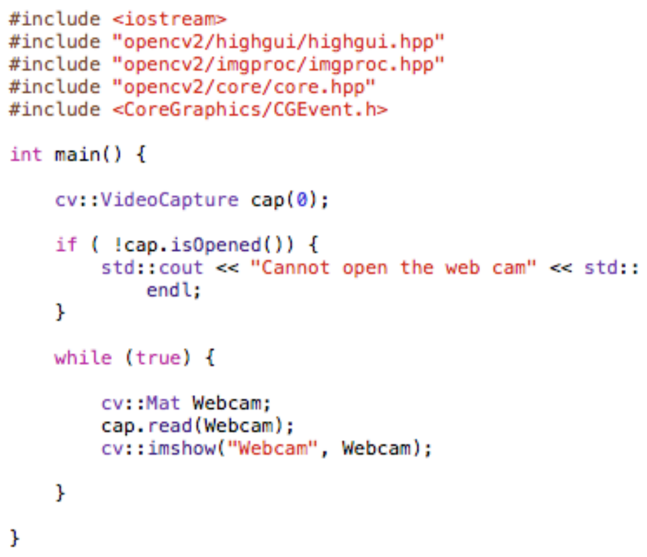
3. After your project finishes loading, take a look at the default assets. The file named "main.cpp" is the file where you will be writing all of your code.
4. Next, we must setup the framework search paths. To do this click on the Xcode project and switch to the Build Settings tab. Scroll to the "Search Paths" subtab, and expand it. From there you must configure "Always Search User Paths" to YES, "Framework Search Paths" to "usr/local/lib", "Header Search Paths" to "/usr/local/include", and "Library Search Paths" to the location of your built OpenCV library (in my case it was "/Users/Timmy/Programming/OpenCV/opencv-2.4.9/build/lib)



▼ Search Paths

Setting	test
Always Search User Paths	Yes ◇
Framework Search Paths	/usr/local/lib
Header Search Paths	/usr/local/include
Library Search Paths	/Users/Timmy/Programming/OpenCV/opencv-2.4.9/build/lib
Rez Search Paths	
Sub-Directories to Exclude in Recursive Searches	*.nib *.lproj *.framework *.gch *.xcode* (*) .DS_Store CVS .svn .git .hg
▶ Sub-Directories to Include in Recursive Searches	
User Header Search Paths	

5. Now, we must add the default system frameworks. For ease of adding, we should first organize out Finder. Launch Finder and click Go > Go To Folder. This will launch a search bar in which you should type "/System". Then drag the "System" folder to the "Favorites" sidebar tab.
6. Right click on your Xcode Project and select "Add projects to [projectname]". Then click System > Library > Frameworks > CoreGraphics.framework and click Add. This framework adds CGEvents which make it possible for us to simulate a key stroke and thus control a game. You may need to add other frameworks the same way, depending on other features, so make sure to consider that.
7. Now we need to add our OpenCV frameworks. Similarly to the system frameworks, you right click on your Xcode Project and select "Add projects to [projectname]". Then find the folder where you built OpenCV, from there click opencv-2.4.9 > build > lib. From there you will see a lot of files ending with .dylib, each controlling a separate function of OpenCV. For our project you must add "libopencv_core.2.4.9.dylib" (core OpenCV), "libopencv_highgui.2.4.9.dylib" (OpenCV GUI control), "libopencv_imgproc.2.4.9.dylib" (OpenCV image proccessing), "libopencv_objdetect.2.4.9.dylib" (OpenCV object detection, not used but helpful to add because it is relevant and may be used later).
8. Before you can reference the contents of these frameworks when coding, you must include them in your script. This will be talked about in the coding section, but is another common source of error.
9. Now lets test to make sure our project is properly built. In "main.cpp" type the following-
10. This will build a new window with a live webcam/Wirecast feed
11. Congrats! You have set up an Xcode project in OpenCV and are ready to begin coding!



```
#include <iostream>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"
#include <CoreGraphics/CGEvent.h>

int main() {

    cv::VideoCapture cap(0);

    if ( !cap.isOpened()) {
        std::cout << "Cannot open the web cam" << std::
            endl;
    }

    while (true) {

        cv::Mat Webcam;
        cap.read(Webcam);
        cv::imshow("Webcam", Webcam);

    }

}
```

<u>OpenCV Syntax</u>

No let's begin discussing OpenCV syntax. After setting all of this up, I'm sure you are anxious to code. Before I started discussing the syntax, I must first note something. There is an irregulatory with OpenCV, and if this syntax does not work it'll give you an error to add "cv::" to all incorrect lines, so make sure to add these lines. This is so because Xcode is trying to compile system C++ code with the OpenCV API's code, so adding "cv::" just distinguishes the syntax as part of the API. I will type some sample code below and then discuss it-

```
//These include the frameworks into this specific script.  The first one is a system reference,
the second one is a system framework reference (for simulating keystrokes), and the rest
reference the .dylib files included previously for OpenCV.  If you want to use a dylib, you must
first add it to th script like so.
#include <iostream>
#include <ApplicationServices/ApplicationServices.h>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/core/core.hpp"

//This so you do not have to add cv:: or std:: in front of different functions to specify where
their syntax is from
using namespace cv;
using namespace std;

//Creates our main function that runs the program
int main()
{
    VideoCapture cap(0); //Capture the video from webcam (location 0), thus running off of the
Wirecast stream

    if (!cap.isOpened()){  //If it cannot find the webcam, print so in log (do not close in case
it skips a frame)
        cout << "Cannot open the web cam" << endl; //Print 'Cannot open the web cam' in the log
    }

    //Creates two windows, one for the HSV or binary image left side, and another for the right.
This windows will be empty bars until we add the rest of the code
    namedWindow("HSVLeft", CV_WINDOW_AUTOSIZE);
    namedWindow("HSVRight", CV_WINDOW_AUTOSIZE);

    CGEventRef wup, wdown, sup, sdown; //Creates CGEvent reference variables for each key, part
of Application Services framework
    //Use the variables declared above to assign the specific key to the CGKeyCode (list if
CGKeyCodes for different keys- http://web.archive.org/web/20100501161453/http://
www.classicteck.com/rbarticles/mackeyboard.php ).  For key press, set it true, for key release,
set it false
    wup = CGEventCreateKeyboardEvent(NULL, (CGKeyCode)13, true);
    wdown = CGEventCreateKeyboardEvent(NULL, (CGKeyCode)13, false);
    sup = CGEventCreateKeyboardEvent(NULL, (CGKeyCode)1, true);
    sdown = CGEventCreateKeyboardEvent(NULL, (CGKeyCode)1, false);

    //While the webcam is on it will call this loop
    while (true) {

        Mat image; //creates OpenCV matrix named image, essentially the webcam feed
        cap.read(image); //reads the matrix 'image' to be processed
        Mat HSVleft; //creates OpenCV matrix named HSV for the left binary images
        Mat HSVright; //creates OpenCV matrix named HSV for the right binary images
        Mat left; //creates OpenCV matrix named HSV for the binary image of the left side
        Mat right; //creates OpenCV matrix named HSV for the binary image of the right side

        //creates OpenCV matrix that takes a certain part of the 'image' matrix, format is- (x
location, y location, x width, y width) and the x/y location are in the bottom left corner of the
crop
        Mat leftimg = image(cv::Rect(0, 0, 640, 720));
        Mat rightimg = image(cv::Rect(640, 0, 640, 720));

        //Left Processing
        cvtColor(leftimg,HSVleft,CV_BGR2HSV); //converts 'leftimg' from BGR (RGB) to HSV (hue,
saturation, value) and assigns it to HSVleft
        inRange(HSVleft, Scalar(0, 0, 100), Scalar(0, 0, 255), left); //creates binary image of
HSV by the range of the first scalar to the second (format is (hue, saturation, value)) and
assigns that to matrix 'left'
        int leftpixel = countNonZero(left);
        imshow("HSVLeft",left); //shows matrix 'left' in window 'HSVLeft'

        //RightProcessing
        cvtColor(rightimg,HSVright,CV_BGR2HSV); //converts 'rightimg' from BGR (RGB) to HSV (hue,
saturation, value) and assigns it to HSVright
        inRange(HSVright, Scalar(0, 0, 100), Scalar(0, 0, 255), right); //creates binary image of
HSV by the range of the first scalar to the second (format is (hue, saturation, value)) and
assigns that to matrix 'right'
        int rightpixel = countNonZero(right);
        imshow("HSVRight",right); //shows matrix 'right' in window 'HSVReft'

        //Keypress Logic
        if (leftpixel > rightpixel) { //if there is more pixels of a color range on the left side
            CGEventPost(kCGSessionEventTap, sdown); //release s key
            CGEventPost(kCGSessionEventTap, wup); //press w key
        }
        else{ //if there is more pixels of a color range on the right side
            CGEventPost(kCGSessionEventTap, wdown); //release w key
            CGEventPost(kCGSessionEventTap, sup); //press s key
        }
    }
    return 0; //ends program if loop does not happen
}
```

If you copy the following code into the previous project, you will see it work. Feel free to comment out and change the code to explore what each function does even further. Also googling is an excellent way to learn about different features and apply them to your game.

<u>Integrating with Unity3D</u>

Once you have finished your Xcode programming, it is smooth sailing to get it to work with unity. When you programmed your key events in the Xcode project, simply use KeyDown in Unity with the same key you programmed to and they will integrate. Simply create your game, and code the player controller to be compatible with that of the OpenCV program. Then, launch the OpenCV program, switch tabs to the game, and watch her run. A suggestion is once you are done prototyping and designing your OpenCV program, take out the “imshow” lines to reduce system lag. Thus, the OpenCV program will still run in the background (there will be no monitoring the OpenCV program), and the game will be much less laggy.

Tweaks and Setting Up the Crowd

For crowd control, we considered a few options. The first was to make double sided paddles, one side green and one side red. However, we ran into a problem as the shadows cast on the paddles made the colors incompatible with the OpenCV program. To fix this we could either shine a bright light at the crowd, or change our approach. Since blinding the audience didn't seem like a good idea, we decided that we needed a self-illuminating source. We tried phones, but not everyone has a smartphone, and then we moved to glow sticks. Glow sticks are the perfect blend, as they are cheap, bright, self-illuminating, and colorful. Thus, we decided to order 1,500 of green and red glow sticks, leaving around 300 of each to test before the assembly. Each student will get one green and one red glow stick, one for each hand, and they will raise the specific hand to interact in a specific way. Before the game can start, we must first calibrate the HSV values to that of the glow sticks. Once you do that, and the green or red glow sticks are the only images on the screen, the game is ready to begin. Save those values and plug them in to the code permanently. You should put the glow sticks under each seat in advance, so the students don't prematurely illuminate them, and make sure to emphasize that they cannot throw the glow sticks. When the game is about to start, make sure all of the lights are off for best performance. Then enjoy, you have successfully created a crowd participation project!