Final Report

Game Development Project

sean khanna – Q11279516

Level 6

Contents

[What? 3](#_Toc7608039)

[How? 3](#_Toc7608040)

[Haar-Cascades 3](#_Toc7608041)

[Local Binary Pattern 3](#_Toc7608042)

[Market 4](#_Toc7608043)

[Goals, Objectives and Processes 4](#_Toc7608044)

[Why some goals might not have been met? 4](#_Toc7608045)

[What I could do next time to complete those goal 5](#_Toc7608046)

[If I was to do it again what changes to my schedule would I make 5](#_Toc7608047)

[What went right and what went wrong? 5](#_Toc7608048)

[Right 5](#_Toc7608049)

[Wrong 6](#_Toc7608050)

[Improvements 6](#_Toc7608051)

[Improvements made by request of client 6](#_Toc7608052)

[Tracking 6](#_Toc7608053)

[Tools 6](#_Toc7608054)

[Future Prospects 6](#_Toc7608055)

[What did I learn 7](#_Toc7608056)

[What could I have done differently during development 7](#_Toc7608057)

[References 8](#_Toc7608058)

[Appendix 9](#_Toc7608059)

[Appendix A – HacknPlan 9](#_Toc7608060)

[Appendix A.1 – HacknPlan Backlog 9](#_Toc7608061)

[Appendix A.2 – HacknPlan Activity Logs (per sprint) 10](#_Toc7608062)

[Appendix B – GitHub 12](#_Toc7608063)

[Appendix B.1 – GitHub Commits 13](#_Toc7608064)

[Appendix C – Black Box Testing 13](#_Toc7608065)

[Appendix D – White Box Testing 13](#_Toc7608066)

[Appendix E – CPU and GPU transfer 14](#_Toc7608067)

[Appendix E.1 CPU 14](#_Toc7608068)

[Appendix E.2 GPU 14](#_Toc7608069)

[Appendix F – Timeboxes 14](#_Toc7608070)

[Appendix G – User Stories 16](#_Toc7608071)

# What?

This project started out as a new tool for facial recognition detection, were I would use certain algorithms and have ways in which the user or developer could manipulate and image or frame with filters and other effects. However, it turns out that creating such a tool meant that the project size was just too great for the time frame allowed. Therefore, the projects goal altered slightly, instead of creating a facial recognition tool, a premade tool was used and applied to the own program.

This projects scope allowed enough time to learn the new library and apply some of its features for a tech demo, with prospects for future use in actual games and apps.

# How?

The project uses a library called OpenCV which is a pre-made library with all sorts of functions methods to manipulate and detect features. For its face detection, the Local Binary Pattern (LBP) was employed rather than Haar-Cascades; the Local Binary Pattern is three times faster at doing calculations because it uses integers rather than floats which Haar-Cascades do. However, Haar-Cascades can be more accurate than the Local Binary Pattern but, for real time detection the Local Binary Pattern surpasses Haar-Cascades.

On the other hand, Haar-Cascades was used to apply eye detection as there was no trained data using the Local Binary Pattern nevertheless, I have prospects for new trained data which will be spoken about later in this report.

## Haar-Cascades

Haar-Cascades simply use an XML document that has been trained for a specific feature with a set of negative and positive images and with these you can input an image into this classifier, superimposing it to the image a return the features locations.

Figure 1 Haar-Cascades, 2019

## Local Binary Pattern

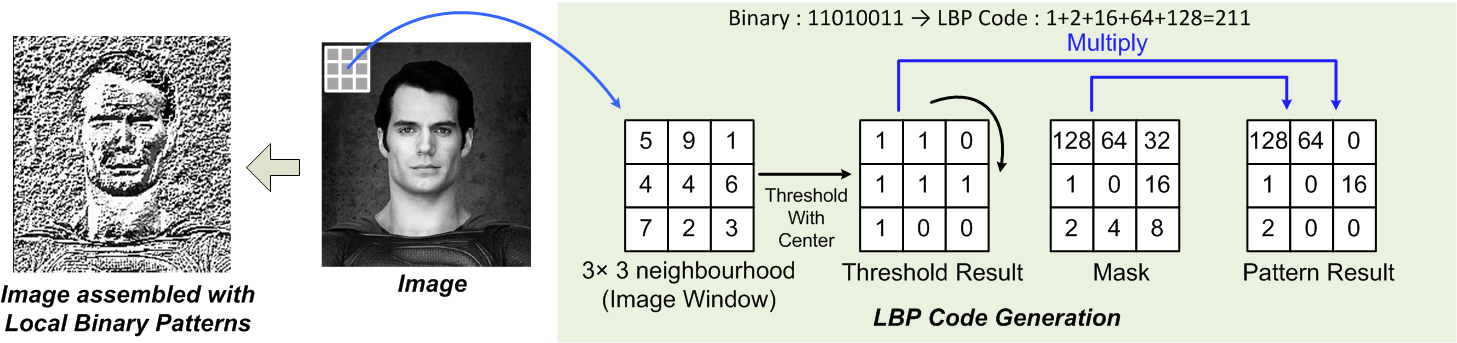
The local binary pattern begins by converting the greyscale image into an image using binary. The first step is to take a 3x3 neighbourhood and take the centre pixel as the threshold, with this you can convert the surrounding pixels into 1s and 0s; apply the mask; eventually you will have an image consisting with multiple binary numbers which can then be used to extrapolate features such as a face or eye outline using histograms.

Figure 2 Local Binary Pattern, 2019

# Market

This project aims to be a program, that people with disabilities can use, just by using there face and eyes, simply by using a normal web camera rather than, for example, purchasing a camera from [tobii](https://www.tobii.com/) (2019), making this whole experience a lot cheaper.

# Goals, Objectives and Processes

Within this project there was only several milestones:

**1st Milestone:** Installing and Testing OpenCV

* 1st step: Download OpenCV
* 2nd step: Install OpenCV
* 3rd step: Test program by writing simple code to load Camera

**2nd Milestone:** Setup GitHub and GitLFS

* 1st step: Create a GitHub repository
* 2nd step: Install GitLFS to allow large file sizes to be uploaded to GitHub
* 3rd step: Upload all files to GitHub

**3rd Milestone:** Implement image filters

* 1st step: Research filters needed to assist with facial detection
* 2nd step: Implement filters and organise them for maximum efficiency

**4th Milestone:** Implement Face Detection Algorithms

* 1st step: Research algorithms
* 2nd step: Implement algorithms
* 3rd step: Test them and then change settings on the filters to help detect the face

**5th Milestone:** Implement Eye Detection Algorithms

* 1st step: Research Algorithms
* 2nd step: Implement Algorithms

**6th Milestone:** Organise Code into classes

* 1st step: Create classes
* 2nd step: Transfer code

**7th Milestone:** Optimise

* 1st step: Change all the MATs to UMATs, which transfers the calculations made on the image from the CPU to the GPU
* 2nd step: Use OpenCL to transfer detection algorithms from the CPU to the GPU

## Why some goals might not have been met?

One of the goals for this project was to have the ability of rotating your head and something on screen happening as a result. Unfortunately, this proved to be very difficult to implement therefore, it was decided to be made in future versions. A simple method for this to work is having the program know where the ears are and when they are hidden. So, if for example, you can see the left ear and not the right, the user must be turning their head right and vice versa. A second option to solve this is to use solvePnP(). Simply this algorithm, detects point features based on a 3D model passed in, say a head. With this information you can work out distances between, for example, the left eye and the nose and the right eye and the nose horizontally, to work out if the tip of the nose is closer to the left eye or right, telling us if they turned their head right or left. As you can see from figure 3, you can perform other things as well like if the user moves their head up or down, giving very accurate information.



Figure Satya Mallick, 2016

Another goal we attempted was to train a new set of cascades for more accurate results, this process is fairly simple, firstly you take around 1000 negative images which doesn’t have the object in, around 20 positive images of the object we do want to detect and then create a descriptor file explaining where the location of the object is within those positive images. Next you have to convert all those negative and positive images into a single vector of images, where the positive images of the object, is put over a set of about 2000 negative images randomly placed, rotated and then cropped. The final step is to run a simple command in OpenCV that takes the 1000 negative images and the 2000 positive images and an XML file that we can use for the detection.

### What I could do next time to complete those goal

If there was a second iteration of this project, it would be a lot easier as the level of knowledge now compared to what it was at the beginning has drastically improved therefore, a lot more time of this project could have been spent learning how to train cascades and how to implement such things as solvePnP().

### If I was to do it again what changes to my schedule would I make

# What went right and what went wrong?

## Things that went right

* After so much effort parts of the program’s workload were shifted from the CPU to the GPU
* Face and eye detection working correctly

## Things that went wrong

* Still have parts of the project that induce lag and slow down the rest of the program
* Detection wasn’t very accurate nor did scenery changes help. The program was unable to adjust in different conditions
* Didn’t utilise most of the filters that were available
* The project was not able to use self-trained data, just the default data

## Improvements

# Improvements made by request of client

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Request | Success or Fail? | Evidence | Notes |
| 1 | Move workload over to GPU | Success and Fail | See [Appendix E.1](#_Appendix_E.1_CPU) for CPU and [Appendix E.2](#_Appendix_E.2_GPU) for GPU. | Some of the work was easily transferrable to the GPU however, the algorithm calculations are still being calculated on the CPU. |
|  |  |  |  |  |
|  |  |  |  |  |

# Tracking

Before starting this project, a whole set of timeboxes (see [Appendix F](#_Appendix_F_–)) was created to aid with keeping to time frames, allowing for as much time as possible to test and document at every step. These time boxes were formed from the user stories (see [Appendix G](#_Appendix_G_–)) created prior to the project. With these time boxes I was able to create a Gantt chart (see [Appendix H](#_Appendix_H_–)) where you could see the dates of all my tasks. Furthermore, I could also input the same data into HacknPlan (see [Appendix A](#_Appendix_A_–)) and get a better overview look with an interactive style to it and a way to separate each task out into their respective sprint.

Trying to stick to this plan created was a very tough task as the lack of knowledge meant that there was endless possibilities of errors and problems that could be encountered and finding a solution to be a task, in itself. On the other hand, the project did keep on track for the most part unfortunately, not all goals were met, mainly because of fiddling with values to get the best results and again the absence of knowledge on some of the harder features like head rotation and winking.

Many problems were encountered when developing this project, the first was uploading to GitHub and having to install GitLFS so that large files could be uploaded. It requires the use of GitBash which was a new tool, that hadn’t been used yet. Once it was up and running though it seemed fairly easy to use.

Secondly, installing the OpenCV \*.dll and \*.lib files proved to be a task as the documentation was a little outdated, so trying to apply it to newer versions of Visual Studio, proved to be difficult. In the end, it was solved, and the project was on its way.

Another problem encountered, was one that is not 100% solved, applying the filters was not to hard however knowing which filter to use and when to use it, was not an easy job and regrettably, still causes a problem when trying to adjust to new environments. All the values need to be adjusted every time the environment changes, because of lighting issues. This causes many issues when trying to do the detection, as they become less accurate and then need adjusting. This however, in future attempts, will need to be done on the fly as to adjust with lighting changes throughout the day, different scenarios and even consumers having items in the background.

## Tools

Throughout any project backups are always key, making sure you save your work is one thing but, what happens when something goes wrong and you lose all your work, if you don’t have a backup then you must start from scratch. As BullGuard (2019) would put PC security, “Having duplicate copies of your most important information saved in a remote location keeps it safe in case anything goes badly wrong with your computer.”. Therefore, this project uses an open source program called GitHub. A place where it can be swiftly uploaded to, safely and securely, and with a means to get access to previous iterations of uploads, as Laurence Bradford (2019) puts it “…like a cloud for code”. Tracking is extremely important in making sure that a project stays on track however, GitHub doesn’t offer any tracking apart from commits (see [Appendix B.1](#_Appendix_B.1_–)) therefore to further aid the project with time management, HacknPlan was used.

HacknPlan is a great tool to help with any agile style program, it allows the use of sprints and boards; you can set time costs, deadlines and allow the ability to log hours (see [Appendix A.2](#_Appendix_A.2_–)).

# Future Prospects

As I have said previously, this program needs to be optimised greatly, all of the calculations, algorithms, filters, rendering and so on, requires massive amounts of computing power to swiftly return results. In turn, the project managed to be split between the GPU as well as the CPU which helped greatly with the optimisation however, it did not solve the problem. After thorough research, I learned a lot about another template library, widely used by many and most programs, Threading Building Blocks (TBB, 2017), it allows programs to “take full advantage of multicore performance”. Using this library will allow for code to be written easily and which adapts, based on the number of cores the computer has.

Unfortunately, one of the aims for this project was to eliminate the use of extra hardware being bought and just utilise what consumers already own, this however, is proving to be very difficult with the number of optimisations needed to make it run smoothly. This is one of the reasons why such companies like tobii(2019) use cameras built by their company, as they have the hardware and software built into the camera to make the workload on the computers less.

To further assist this project, a profiler will be used to help deduce where and when the CPU and GPU are being used and find ways to better optimise it, allowing for quicker and higher quality results. Visual Studio has a profiler built in with enough documentation (see [here](https://docs.microsoft.com/en-us/visualstudio/profiling/?view=vs-2019)) to help with learning how to use it.

# What did I learn?

One of the main things learned during the development of this project was, how important it is to optimise as much as possible. There are so many ways in which you can optimise a project and for such a project like this, it is essential to optimise as much as possible.

# What could I have done differently during development?

During development there was a lot I could have done differently starting with organisation. At the beginning, it was more about testing to see if things worked the way they should, rather than readability. Inevitably, it got to a point where I had to start tidying things up and creating classes. This took a long time trying to set up all the classes and pass pointers around however, I did manage to optimise the program better and allow it work faster doing so.

# References

tobii, 2019. The world leader in eye tracking [viewed 23/04/2019]. Available at: <https://www.tobii.com/>

OpenCV, 2019. Cascade Classifier Training [viewed on 23/04/2019]. Available at: <https://docs.opencv.org/3.3.0/dc/d88/tutorial_traincascade.html>

Viola P., Jones M., 2004. Object Detection Using a Boosted Cascade of Simple Features [viewed on 23/04/2019]. Available at: <http://www.merl.com/publications/docs/TR2004-043.pdf>

Christos Kyrkou, 2017. Object Detection Using Local Binary Patterns [viewed on 23/04/2019]. Available at: <https://medium.com/@ckyrkou/object-detection-using-local-binary-patterns-50b165658368>

Intel®, 2017. Threading Building Blocks [viewed on 26/04/2019]. Available at: <https://www.threadingbuildingblocks.org/>

BullGuard, 2019. Backup of Data & Files – Why it is Important? [viewed on 26/04/2019]. Available at: <https://www.bullguard.com/bullguard-security-center/pc-security/computer-threats/backup-of-data-files-why-it-is-important.aspx>

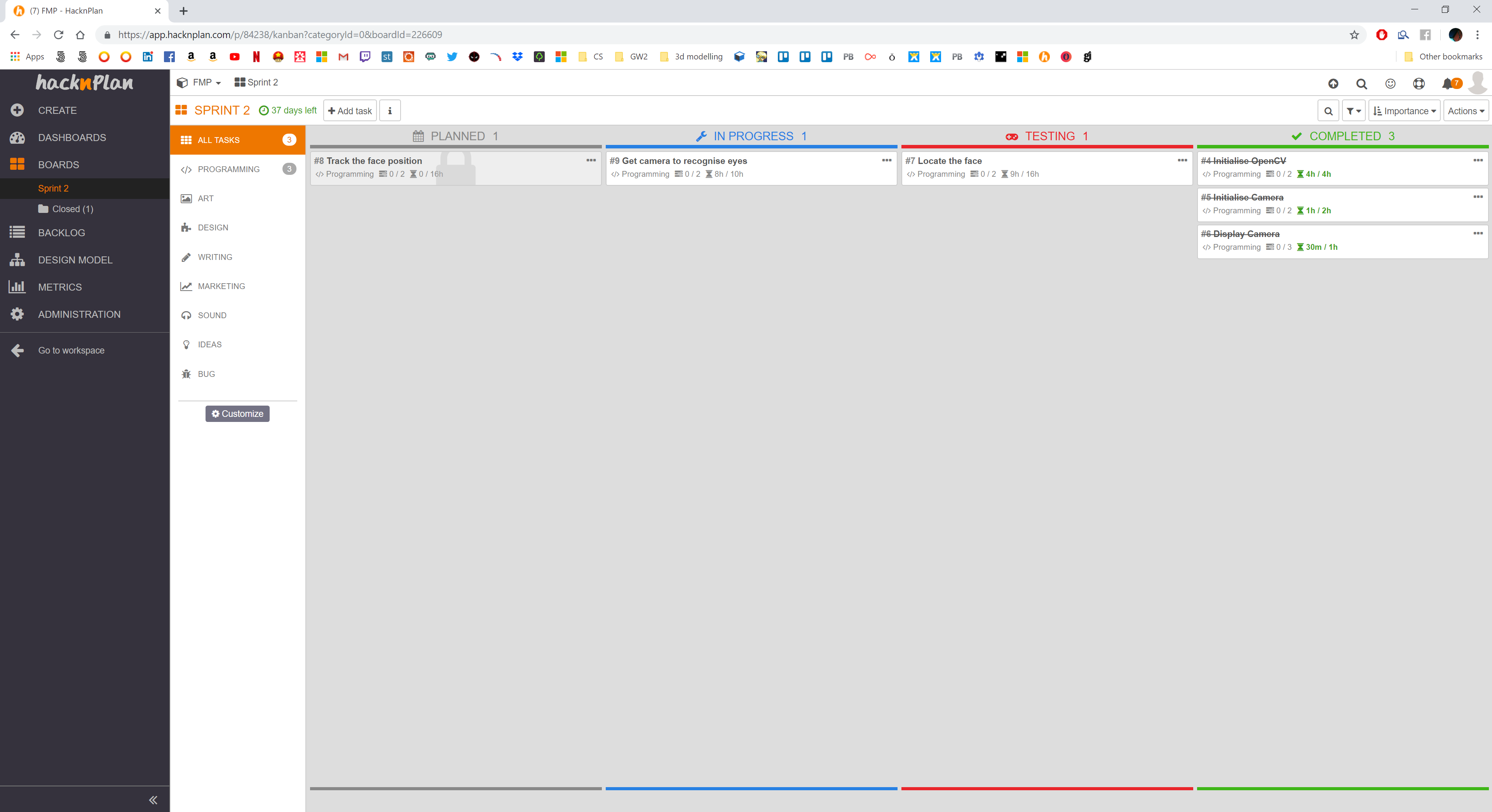
Laurence Bradford, 2019. What Is GitHub, and Why Should I Use It? [viewed on 26/04/2019]. Available at: <https://www.thebalancecareers.com/what-is-github-and-why-should-i-use-it-2071946>

Satya Mallick, 2016. Head Pose Estimation using OpenCV and Dlib [viewed on 29/04/2019]. Available at: <https://www.learnopencv.com/head-pose-estimation-using-opencv-and-dlib/>

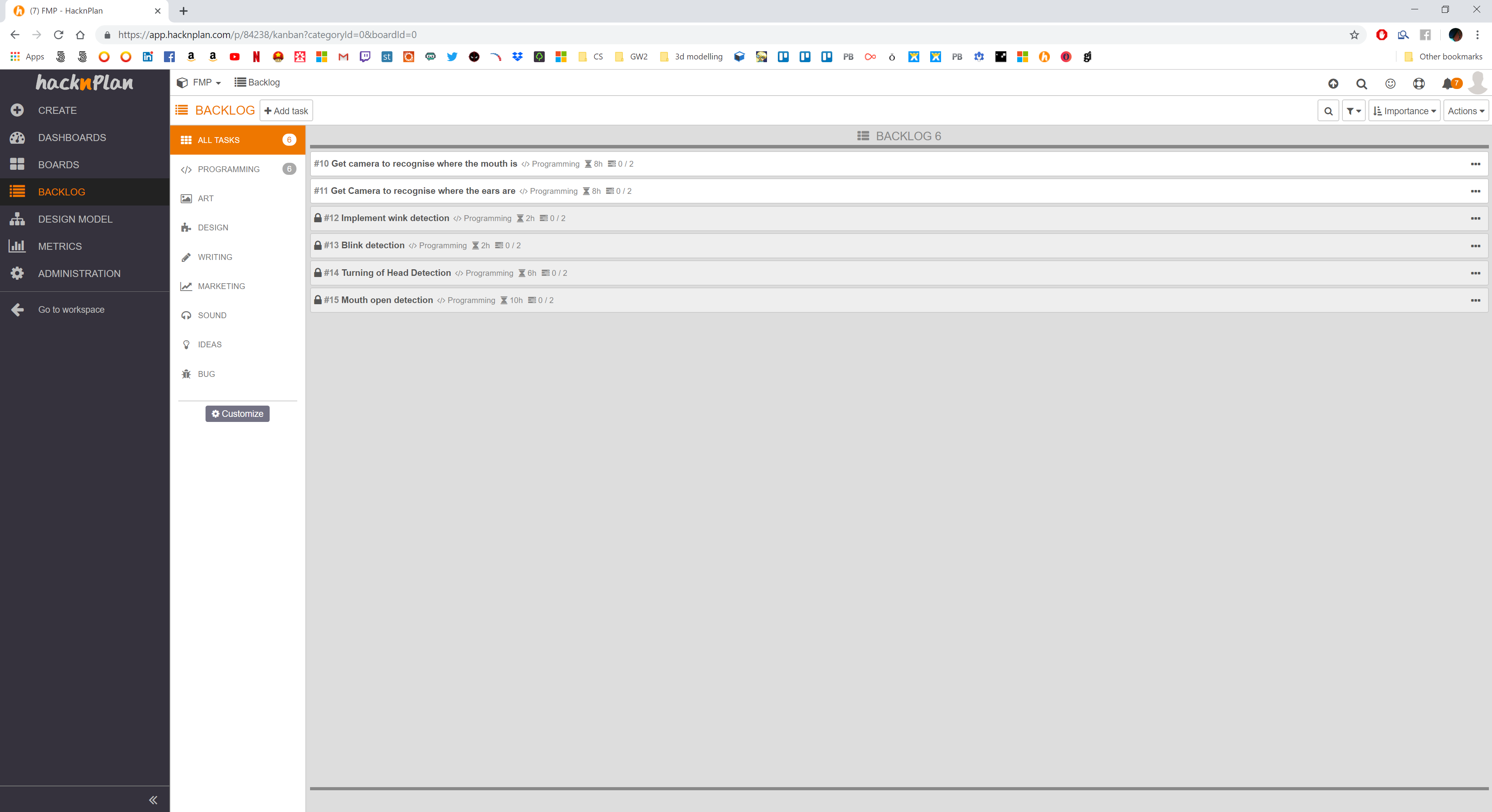
Visual Studio, 2019. Measure app performance in Visual Studio [viewed on 01/05/2019]. Available at: <https://docs.microsoft.com/en-us/visualstudio/profiling/?view=vs-2019>

# Appendix

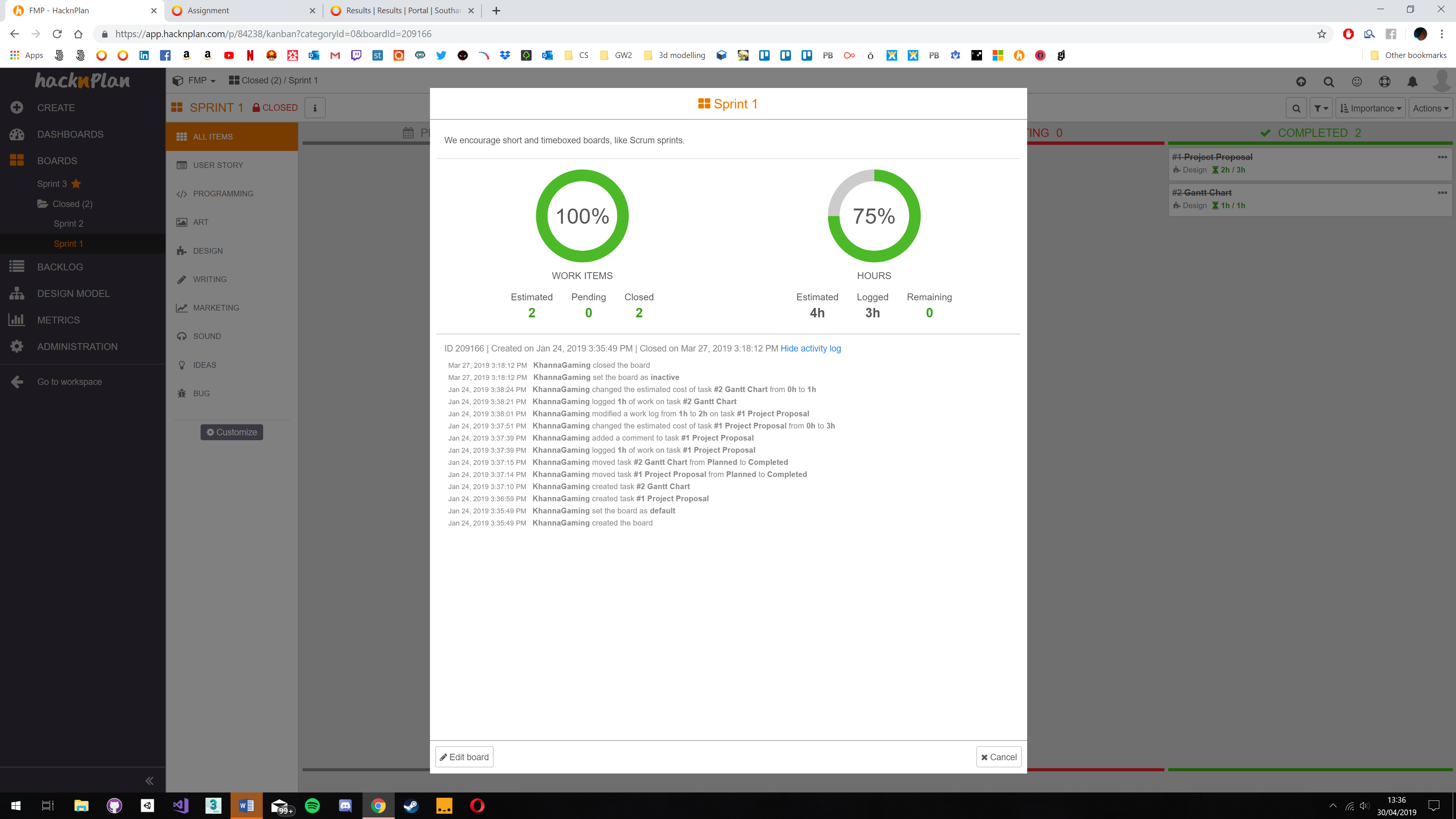
## Appendix A – HacknPlan

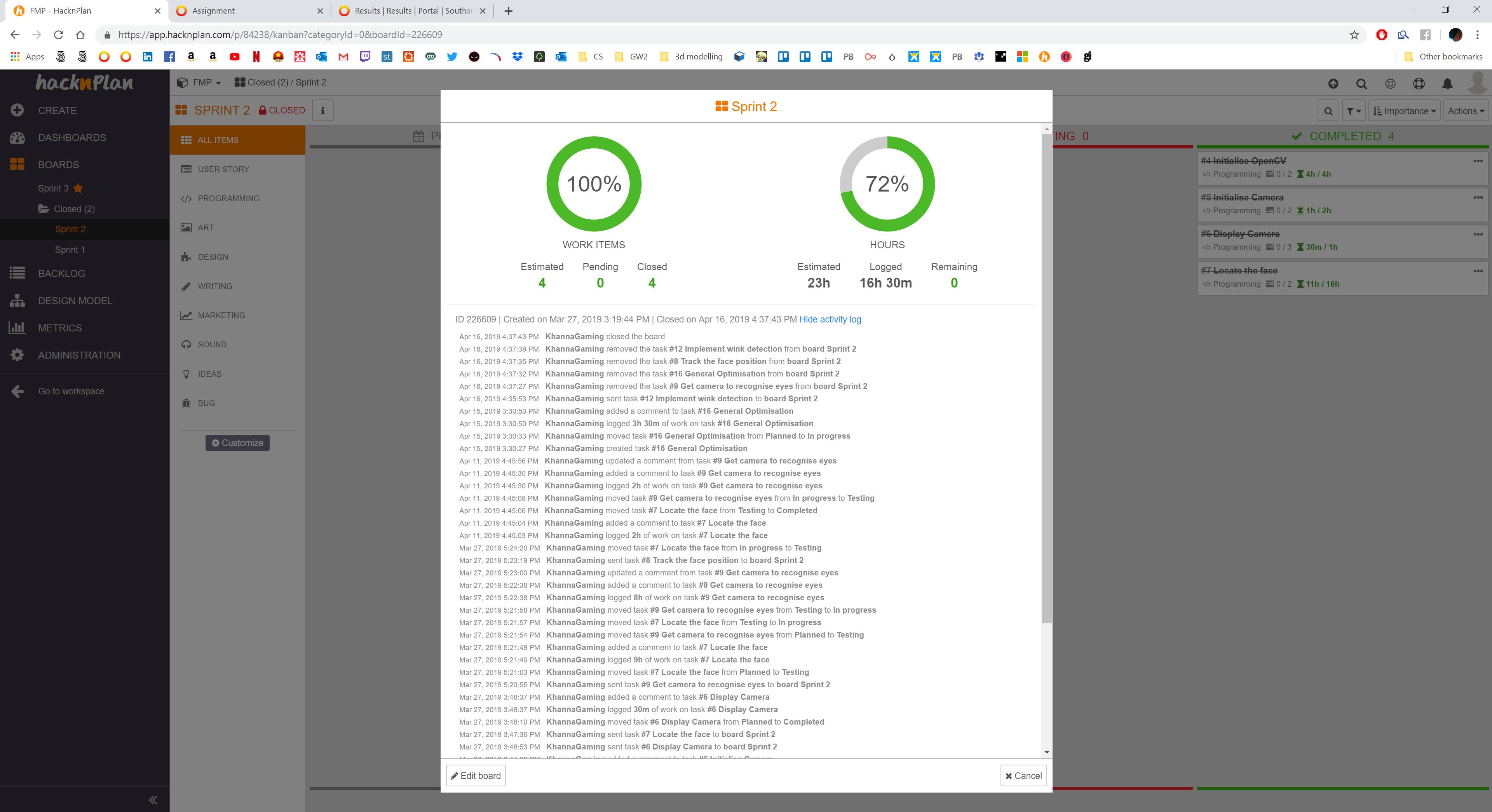


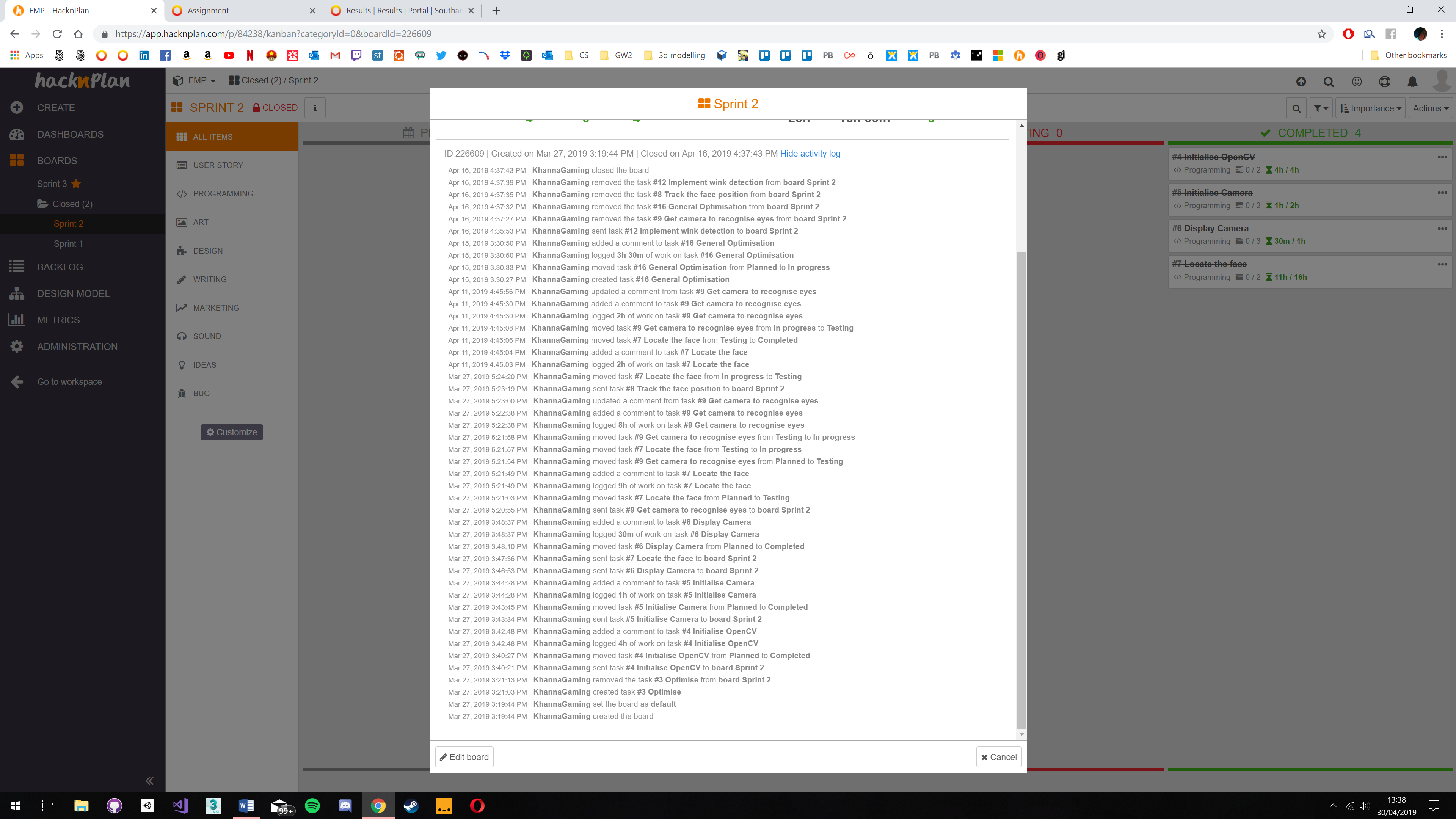
### Appendix A.1 – HacknPlan Backlog

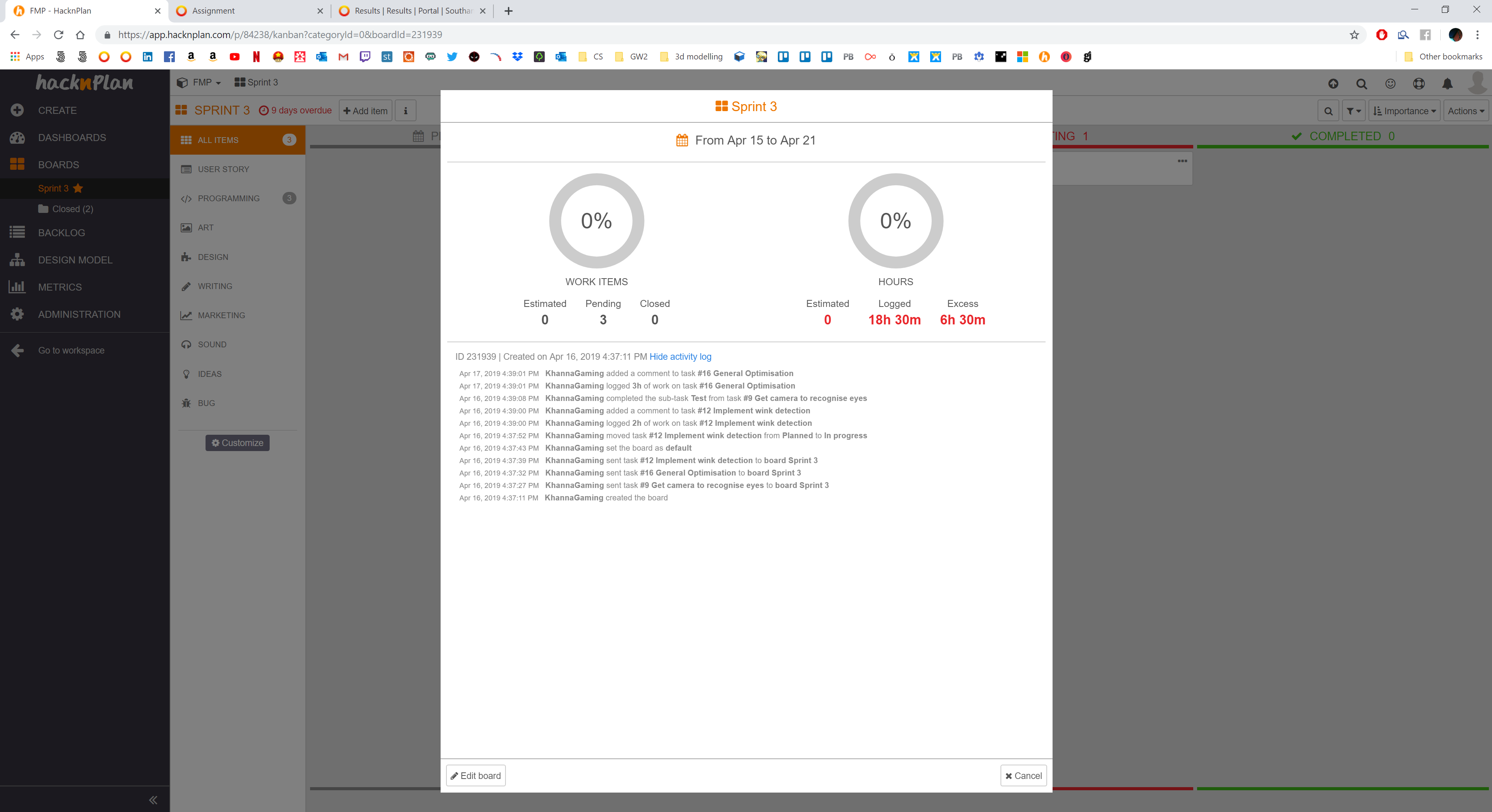


### Appendix A.2 – HacknPlan Activity Logs (per sprint)

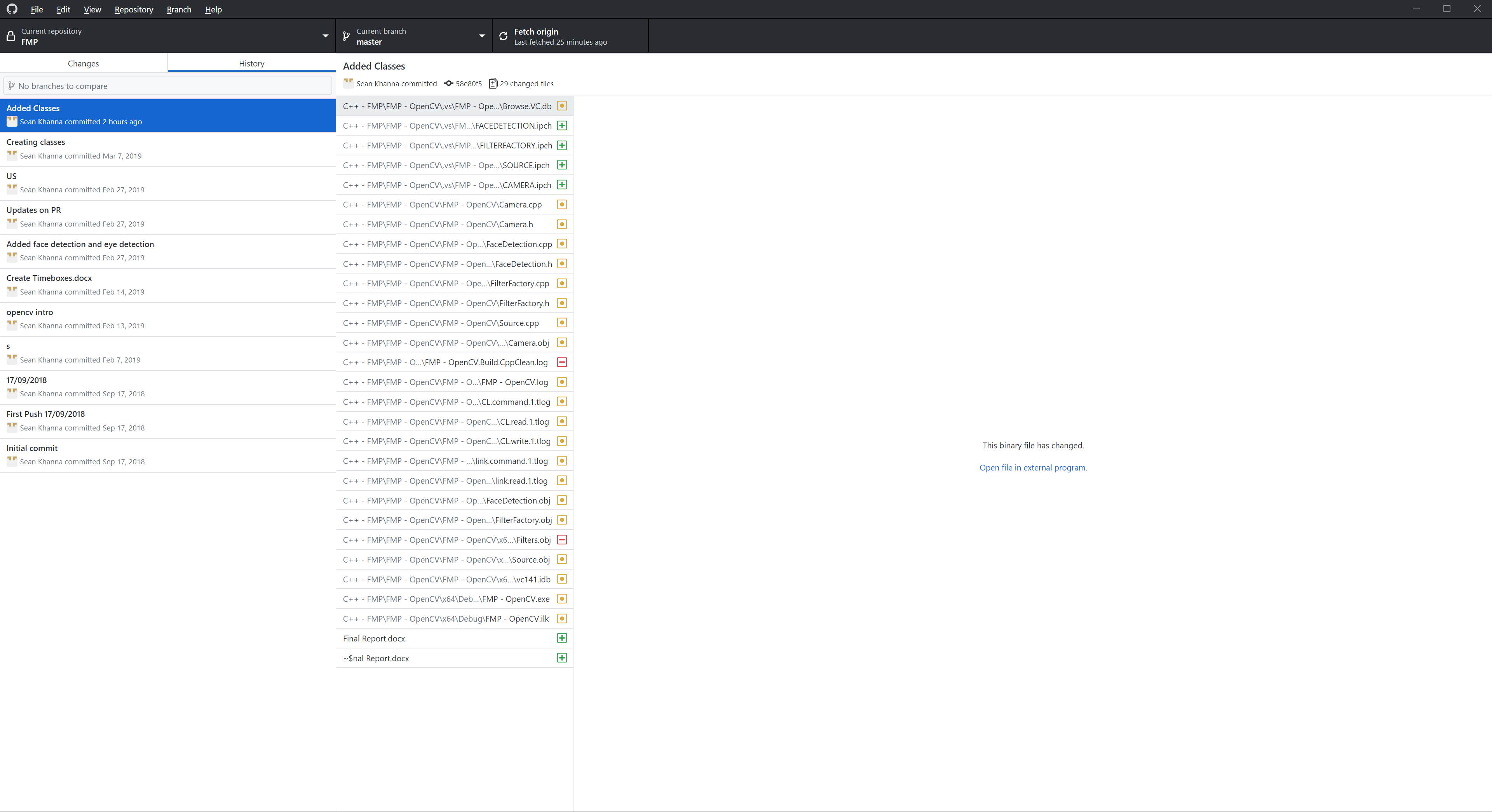




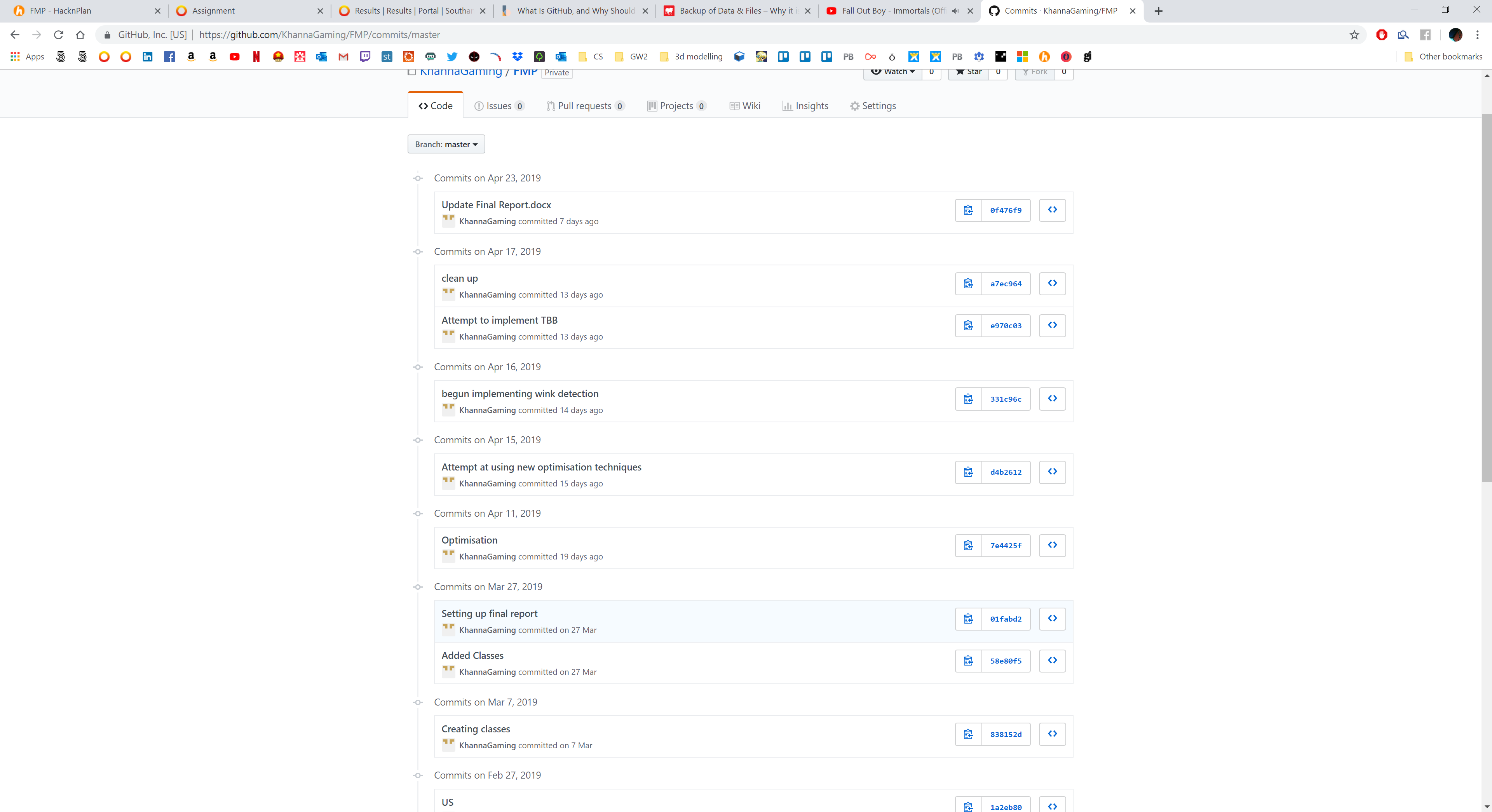


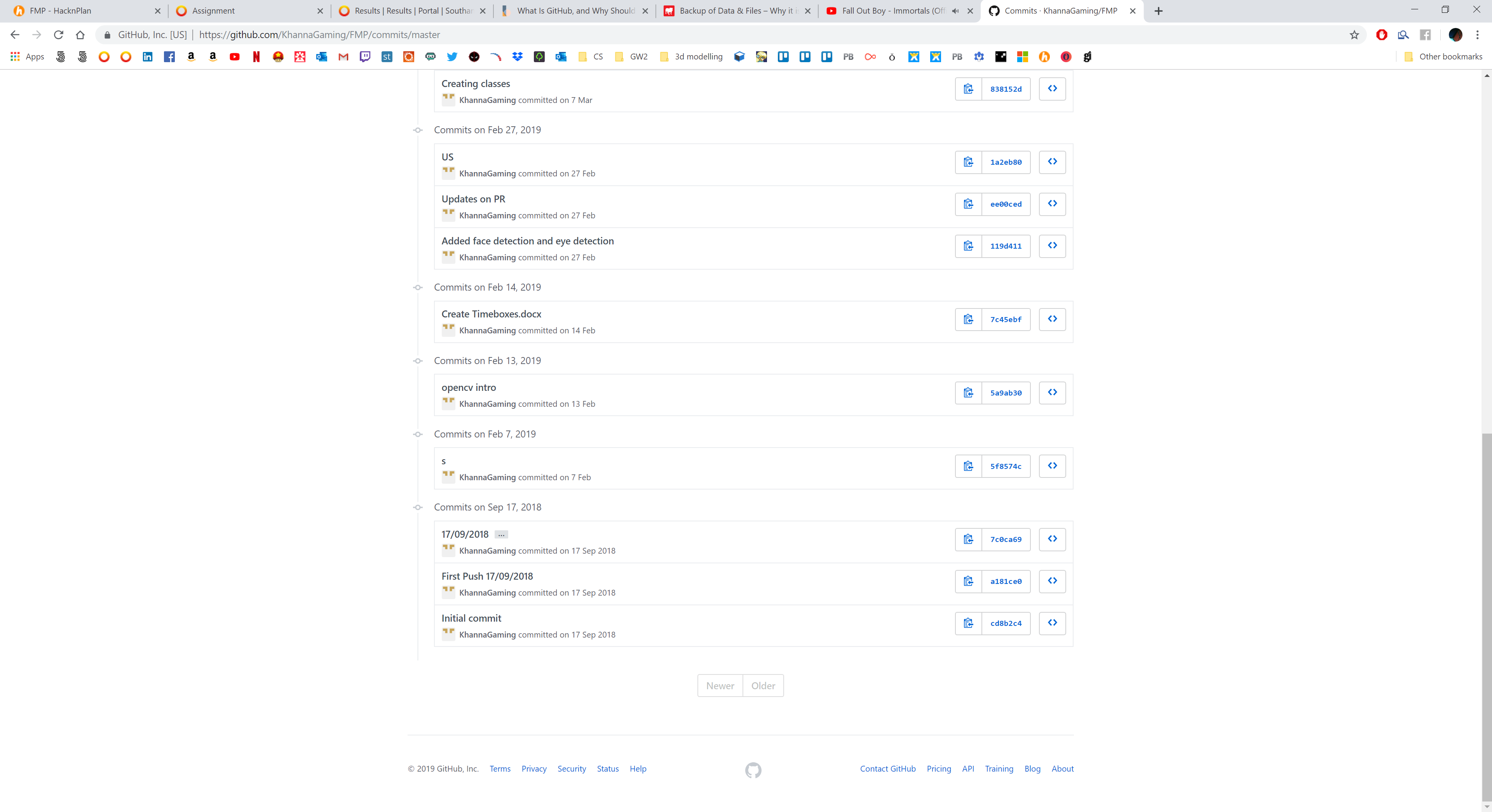


## Appendix B – GitHub



### Appendix B.1 – GitHub Commits





## Appendix C – Black Box Testing

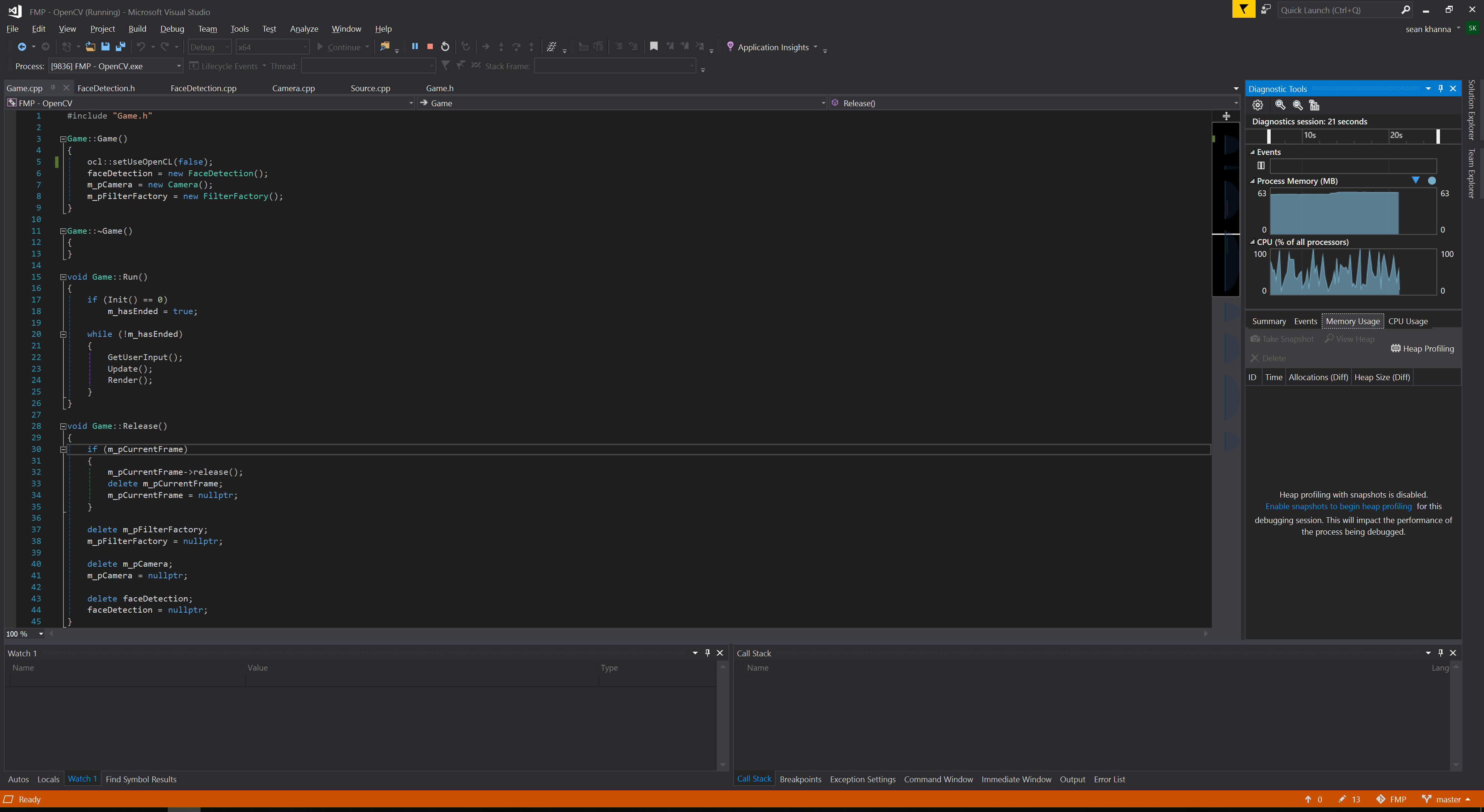
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Feature | Test | Expected | Actual | Solution |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Appendix D – White Box Testing

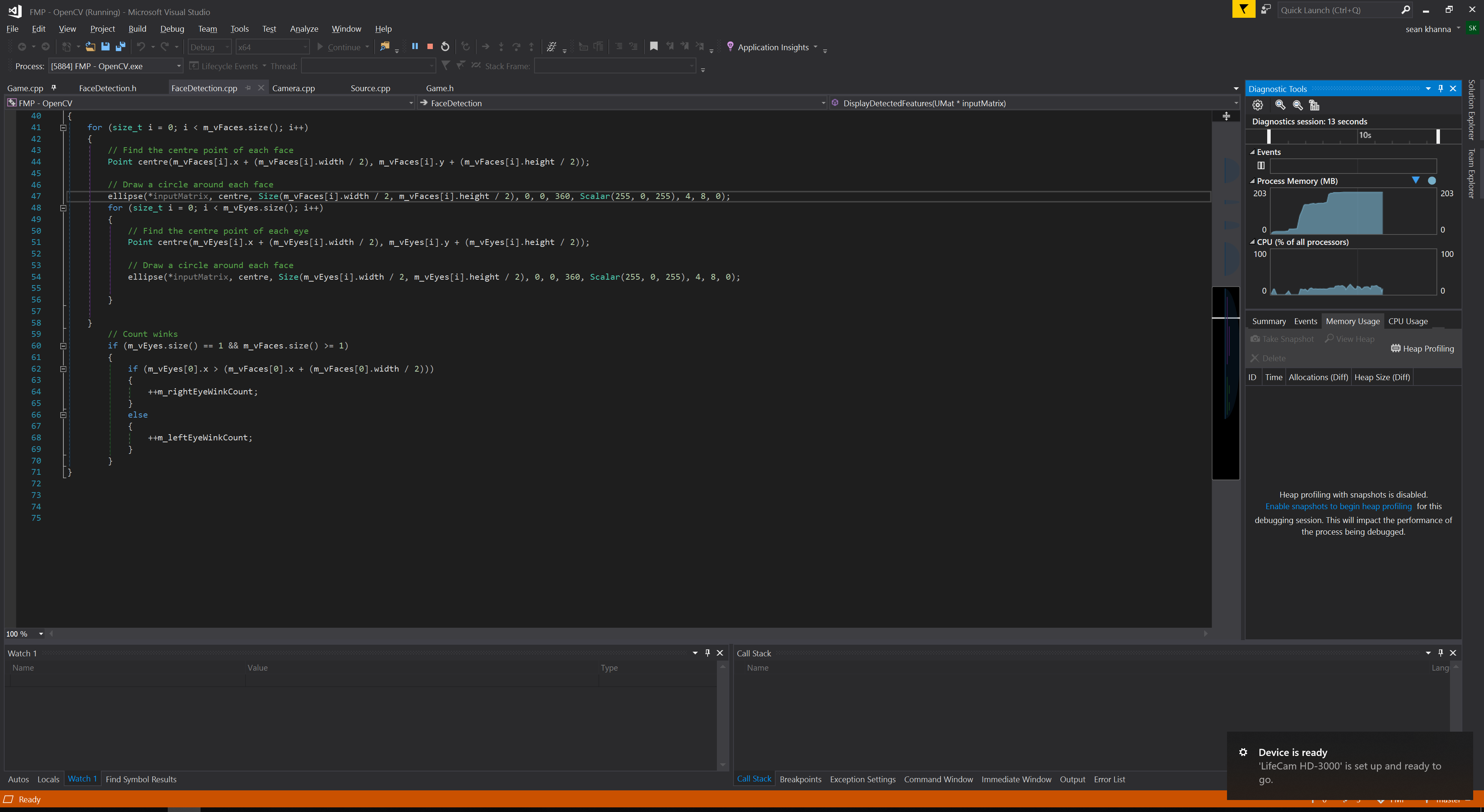
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Appendix E – CPU and GPU transfer

### Appendix E.1 CPU



### Appendix E.2 GPU



## Appendix F – Timeboxes

1. Setting up camera including initialisation of OpenCV - 30 hours
   1. Initialise OpenCV - 20 hours
      * Test it works – 17 hours 30 mins
      * Document – 2 hours 30 mins
   2. Initialise Camera – 10 hours
      * Test – 7 hour 30 mins
      * Document – 2 hours 30 mins
      * Dependencies: 1.1
2. Display camera – 5 hours
   1. Show camera on screen – 5 hours
      * Test – 3 hours 45 mins
      * Document – 1 hour 15 mins
      * Dependencies: 1.2.
3. Getting camera to recognise where the face is – 105 hours
   1. Locate the face – 50 hours
      * Dependencies: 1.2.
   2. Track the face position – 40 hours
      * Dependencies: 3.1.
   3. Testing – 10 hours
   4. Document – 5 hours
4. Getting the camera to recognise where the eyes, mouth and ears are – 90 hours
   1. Eyes - 40 hours
      * Test - 35 hours
      * Document – 5 hours
      * Dependencies: 1.2.
   2. Mouth – 25 hours
      * Test – 20 hours
      * Document – 5 hours
      * Dependencies: 1.2.
   3. Ears – 25 hours
      * Test – 20 hours
      * Document – 5 hours
      * Dependencies: 1.2.
5. Use this to calculate if a player is winking, blinking, turning their head etc – 60 hours
   1. Winking – 10 hours
      * Test – 7 hours 30 mins
      * Document – 2 hours 30 mins
      * Dependencies: 4.1.
   2. Blinking – 10 hours
      * Test – 7 hours 30 mins
      * Document – 2 hours 30 mins
      * Dependencies: 4.1.
   3. Turning head - 15 hours
      * Test – 13 hours
      * Document – 2 hours
      * Dependencies: 3.2.
   4. Mouth open – 25 hours
      * Test – 20 hours
      * Document – 5 hours
      * Dependencies: 4.2.

**Grand total:** 290 hours.

## Appendix G – User Stories

As a player I want to be able to turn my head to make my character move because I want to explore.

As a player I want to be able to use my eyes by winking or blinking to do certain actions because I want to be able to communicate with NPCs.

## Appendix H – Gantt Chart

