Amyleila Mejia

**Conclusion:**

A summary of the final design should be given to conclude the document. Testing data showing that the design meets the problem requirements can be reiterated in this section. Possible improvements to the design should be mentioned if they were not implemented. This section serves to conclude the document and convince the reader that the design successfully meets all problem requirements.

-------------------------------------------------------------------------------------------------------------------------------

For the most part, the project completion & requirements were fulfilled. In the case of Object Detection, Unity uses a built-in package called Barracuda, a neural network interface for Machine Learning Models, ML, offering a production ready system. The team’s first choice in neural networks was to go with the built in ML in YOLO, Darknet. But because Darknet would not communicate between the YOLO algorithm and the Hololens a bridging software was necessary to make communication between these happen. The way YOLO works is that it applies a single neural network to an image then it divides it into regions all while making predictions and probabilities about detection. Once YOLO has been trained it produces a weights file that the built in neural network on YOLO, Darknet, uses for object detection. This approach makes it faster and different to the classifier-based system approach which makes the system slower. The choice to commit to Barracuda despite readily having Darknet was made because Unity had an existing plug-in for Barracuda which would make interfacing with the Hololens an easier task. YOLO still uses Darknet as its main neural network but Unity requires Barracuda.

The first issue that arose as a result of using the Barracuda software was the conversion of the weights file produced by YOLO into an Open Neural Network Exchange, ONNX, file for Barracuda. The ONNX file is an interface file type that stores the model information that Unity can read via Barracuda. This conversion would allow the loading of the Darknet trained neural network into Barracuda as an ONNX file/network. The issue was in the conversion from the weights file into an ONNX file. The ONNX file that was produced was incorrect and had issues and could not be used by Barracuda. The first meeting with Dr. Wortman tried to address these file conversion issues which mainly revolved around loading the ONNX file into Unity. Some errors were fixed, however the underlying problem was that the file was converted incorrectly. Tools to properly convert the ONNX file do not yet exist. This is an observation that was made from the team’s extensive research. Major progress was made with the investment and knowledge provided by Dr. Wortman. Unfortunately the ONNX file conversion still failed. This ultimately did not allow the team to proceed with a successful rendering of the game on the Hololens and adjustments had to be made.

To bypass not being able to render on the Hololens was the production of two different demonstrations to show the working parts of the game . A lot of effort was put into making different aspects of this project to function correctly. The only issue was the collaborative effort of trying to implement the different programs and have them work as a unit. Rendering on the Hololens was one of the main specifications for this project but due to a time constraint pursuing a resolution to this issue would not have allowed enough time to produce demonstrations for the other aspects of the project that did work correctly. Only four specifications failed to meet the status quo. Eleven out of a total of fifteen requirements were met for this project making it a 73.33% success. This was calculated using the Equation (1). Table 1 **(include section reference)** summarizes the findings on the specifications that were met and uses those findings to calculate the overall success percentage.

Equation (1)

In order to demonstrate the other aspects of the project working correctly two demonstrations were created. One showcases the game logic working correctly along with the special effects of audio and graphics. It hardcodes different player and dealer hands and displays the probability of those different plays in real time. Because the neural network file could not be used by unity a simple hard coded scenario was generated showing that the game logic works as it is intended too. The other showcases the trained algorithm working correctly to detect an object. This process uses the Darknet demo test program which is responsible for producing bounding boxes over objects that were successfully detected.

Once committing to the changes in the plan of execution and agreeing on the demo versions new problems arose. The user interface was not drawing the correct text for one of the probabilities. It would skip the first probability calculated, even though it obtained the correct probability and would print it on the debug console. To resolve this bug, the fix is to write to the UI text in Unity’s main thread. To use Unity’s main thread, this github repository was used: <https://github.com/PimDeWitte/UnityMainThreadDispatcher>. Changes to the original game logic were made in order for the logic to be accessed by the main thread. Additionally for the program to work, the full path of the blackjack.py script needs to be specified in the code. So if the program tries to run on a different computer, the path most likely will need to be changed.

***Table 1:*** Result of the findings regarding the parameters and requirements set forth for the project.

| **Requirement & Parameters** | **Pass** | **Fail** |
| --- | --- | --- |
| Implemented using a HoloLens |  | x |
| Total system latency will be under two seconds |  | x |
| Improve runtime and battery life |  | x |
| Perform the app rendering and processing on a host computer |  | x |
| The AR/VR device will connect to a Wi-Fi router in order to communicate to the host computer | x |  |
| Determine if the player should hit, split, or stand for their next move | x |  |
| Determine the value of the cards held by the dealer | x |  |
| Display the probability of success for each move the player could perform | x |  |
| Display a graphic when player hits a card count of twenty-one | x |  |
| Display the statistics for next move that the player should perform | x |  |
| Play a sound when player wins the game | x |  |
| Identify the playing cards that a person is holding | x |  |
| Determine the value of the cards that are given to the player | x |  |
| Display the value of the cards that are on the table | x |  |
| Accompany a set of documents specifying every piece of software and its functionality | x |  |

These changes allowed for the project to demonstrate the majority of the working aspects of the project. Although it was done separately it still holds its 73.33% success and when training the Object Detection, we received a theoretical mean accuracy percentage, mAP, of 100% and an average loss of 0.02. In order for YOLO to function it must display a 0.25 confidence detection rating or higher and in comparison to our rating of 50% from real world tests. It significantly surpasses that rating and successfully detects accurately. Both demonstrations work properly meeting most of the passing requirements. Time constraint had a huge impact on the success percentage of the overall project but the team is confident on the progress and functionality of the project. This project can be handed down to a different group where a higher success rate can be made.