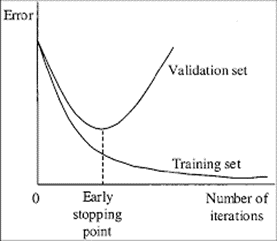
Stefano Signorelli

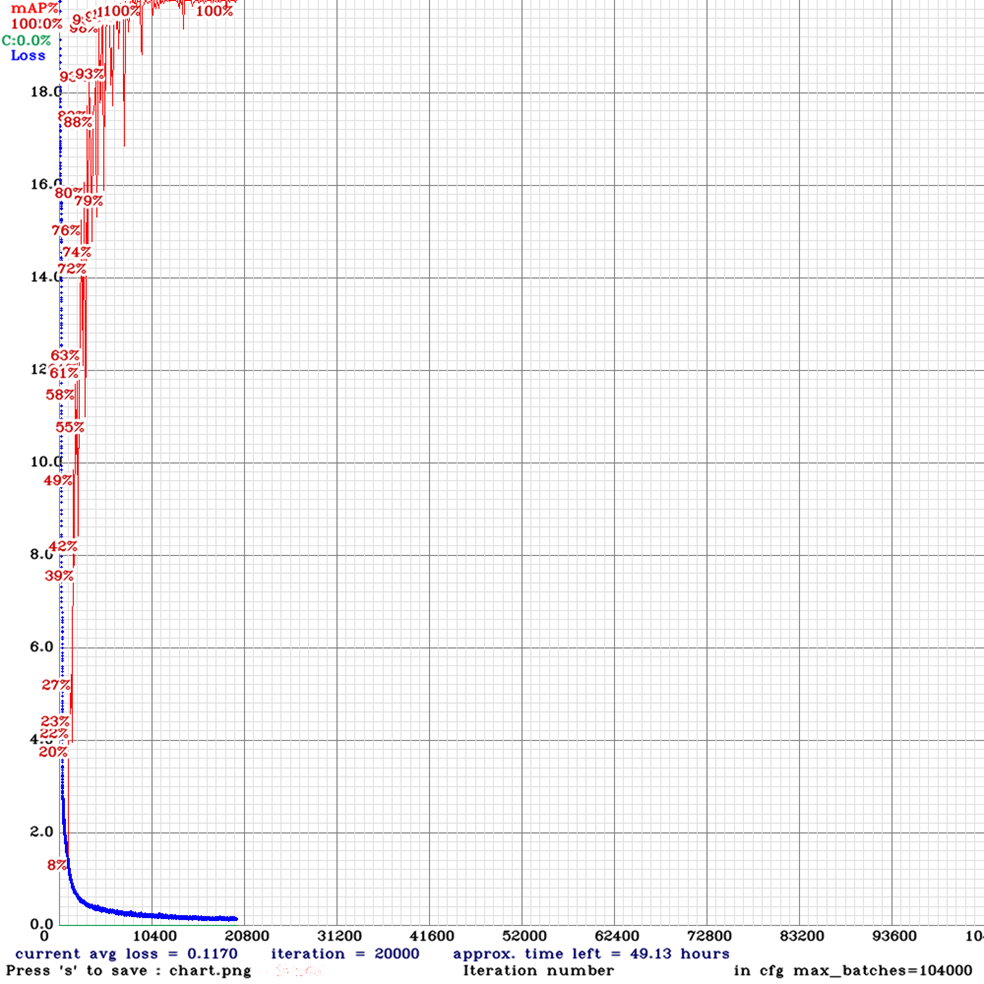
**Results and Analysis**

Data should be presented appropriately, as tables or graphs, whichever is best. All figures and tables must be introduced, captioned, and referenced in the section text. A brief description of the trends should also be given when introducing each figure and/or table. Results should be interpreted in the context of the reason for testing. Averages, standard deviations, confidence intervals etc. should be provided where appropriate. A single result with no uncertainty does not represent thorough testing and will not fill the sponsor with confidence in the product.

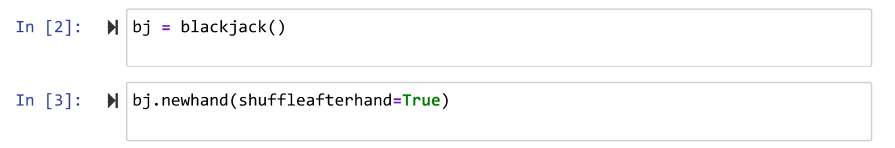
[DONE BY FRIDAY]

[Austin] Put mAP & avg loss graph

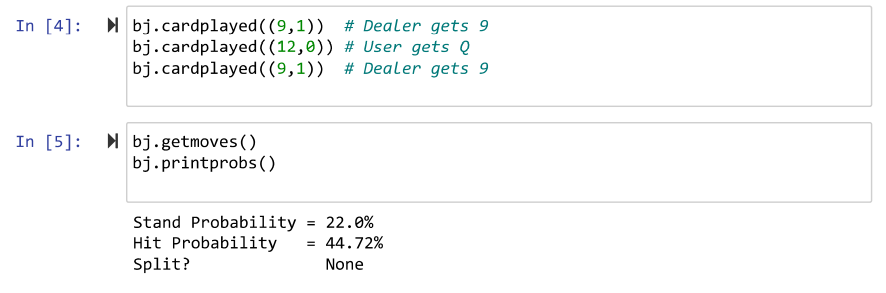
For testing of object detection, we can use metrics from training, as when training the current dataset, we use mAP (mean Average Precision) which describes the perceived accuracy of training by comparing the training dataset to the validation dataset. We also use average loss or error. See the plot below. Ideally, we want a high mAP (accuracy) and a low error. For around the 20,000 iterations we trained for, the below graph is shown of the mAP and avg loss. From the graph, you can see the avg loss ended up being 0.117 & mAP = 100%. We believe that the stopping point chosen, defined as “early stopping point”, can be more accurate than training for 40,000+ iterations.



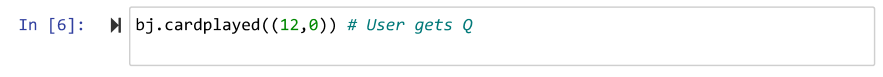
While designing the probability algorithm, a jupyter notebook was created to both demonstrate how to use the class, and test that it works as intended. A jupyter notebook is a feature of python that allows you to run code in ‘cells’, while a kernel runs in the background. The first cell is very long, so it will not be detailed here, but it defines the blackjack class that is used to perform all the necessary calculations.

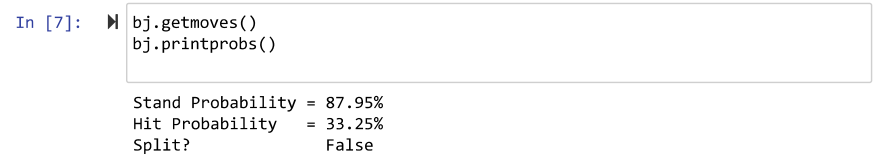


In cells 2 and 3, the class is instantiated, and a game is begun by setting up a new hand.

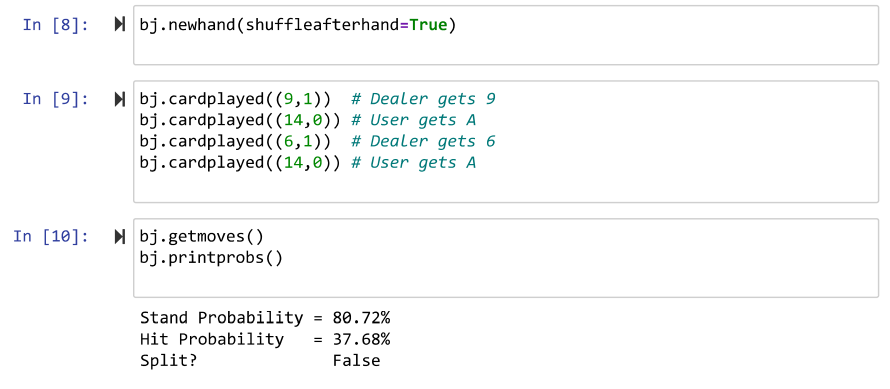


In cell 4, three cards are played, two for the dealer, and one for the user. In cell 5, getmoves is called to do the calculations, and printprobs is called to print them out, as shown.

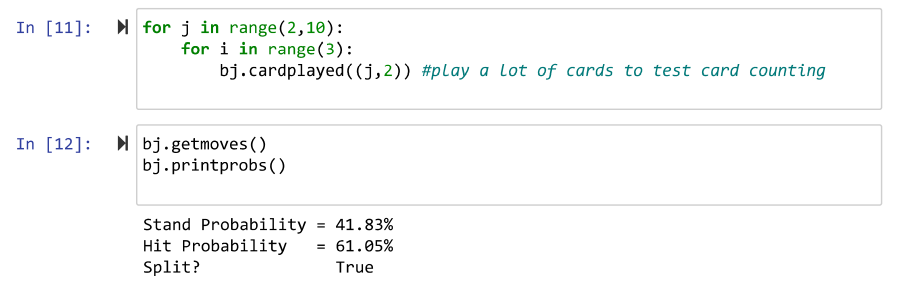




In cell 6, another card is played, and in cell 7, the probabilities are reprinted to show that they have changed as a result of a card being played.



Cells 8, 9, and 10 are similar, but the cards played demonstrate the program deciding whether or not the user should split.



Cells 11 and 12 continue on to show that by changing the cards in the deck, by playing cards to neither the dealer or user, the probabilities are changed so much that splitting becomes the correct choice, where it was not before.

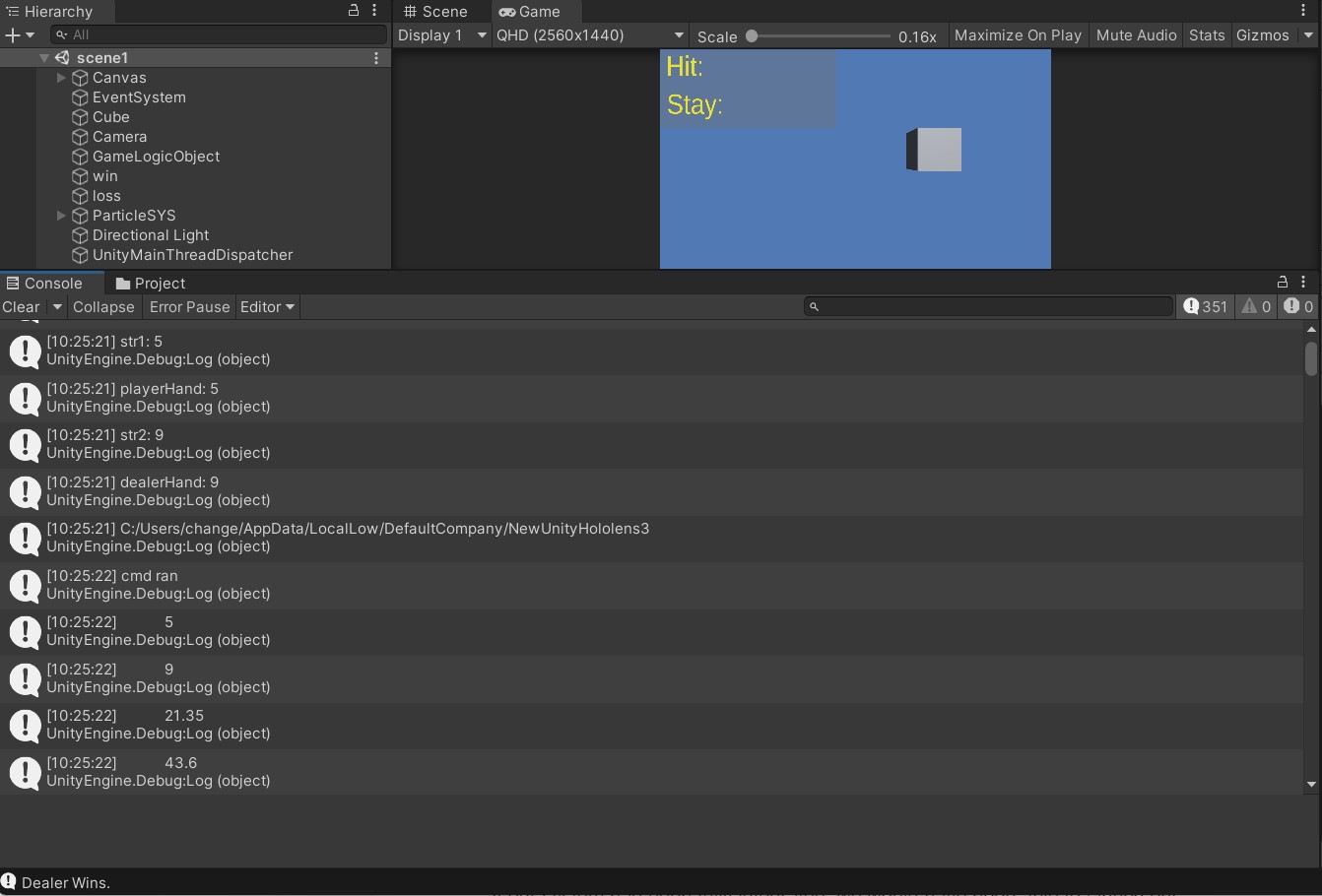
[Stefano] UI steps

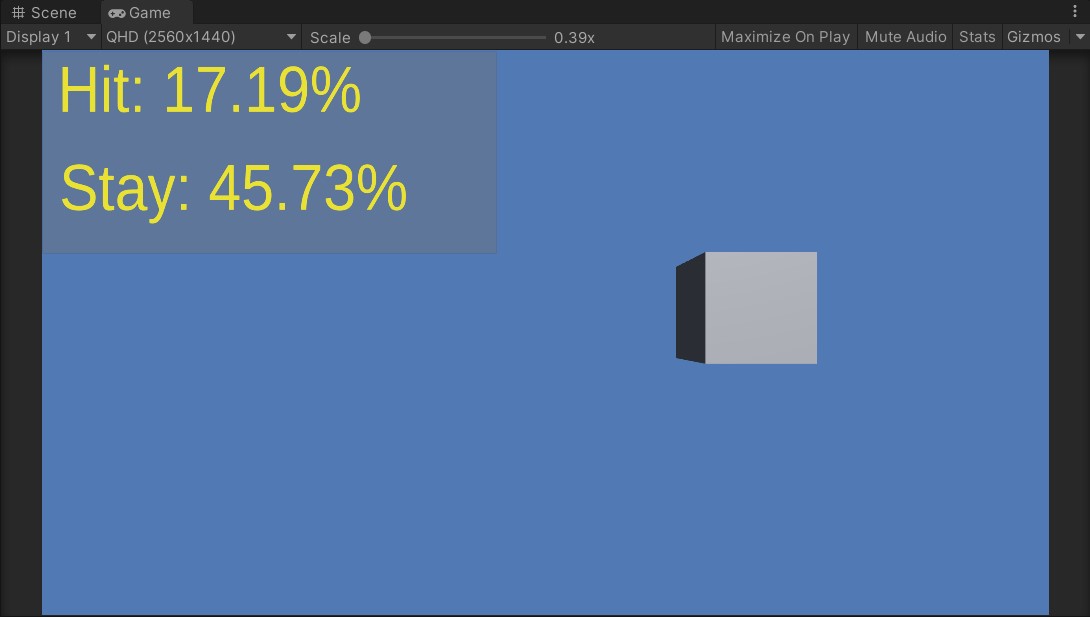
To test the UI, after building the UI solution in the “GameLogic.cs” C# script, the Unity Engine Debug Console was used to print out the probability values to determine that the UI is working correctly. Once it was built correctly, it would print out the values in the UI in yellow text, the “Stay” probability, and the “Hit” probability for the player. This testing using the debug console helped find a solution to a bug that was experienced in the design process. This bug was that it would not display the first probability on the UI text: “TextMeshPro - Text” object, even though it recognized the correct probability when printing it to the debug console. Ultimately the fix to the bug has been found after looking at a forum: "<https://forum.unity.com/threads/textmesh-pro-is-not-updating-when-changing-text-programatically.490125/page-2>”, it mentioned that running the needed code in the main thread of Unity Engine would fix it. To run to the Unity Main thread, we used this Github repository which was linked to the forum found: “<https://github.com/PimDeWitte/UnityMainThreadDispatcher>”.

Overall, the debug console has been crucial for performing constant tests during the development of the UI for the blackjack assist app. It was constantly used to verify that variables and data structures are correct, as well as the logic for the app. For example, the global list: “argumentList” was iterated and printed to show it has the correct values. It has single character strings inside the list which represent the hands for the player and for the dealer. It has “p”, and “d” to denote dealer and player, and then the following card values are specified for player and dealer. This list is used to build the command line command with command line inputs when executing the blackjack.py python script with command prompt.

The following photo shows the debug console, it can be seen it prints the probabilities obtained from the “blackjack.py” file, and as well as single numbers which are from the global list, “argumentList” values. The probabilities seen in the photo are 21.35%, and 43.6%, they don’t have the percentage sign in the debug console, but they are the probabilities for that round of blackjack. Additionally, a notification from the debug console can be seen from the picture, it mentions who is the winner for that round of blackjack.

As a result, the testing performed on the user interface (UI) showed the correct behavior, and displayed the correct probability values obtained from the probability algorithm in the python blackjack script “blackjack.py”. The correct probabilities obtained from “blackjack.py” are reflected in the actual UI when starting the app, as well as on the debug console. Other tests performed, specifically executing the python script on command prompt with command line arguments, a text file was written to show the correct command line argument values has been received successfully from the “GameLogic.cs” C# script. Other successful tests performed include printing to the debug console who is the winner for that round of blackjack, the player, or the dealer. Testing to check that the global lists had the correct values from the set values for the player and dealer hands has been tested using the debug console, and is working correctly.





[Amber] Mention traces we would have gotten

If our system had been fully integrated, we would have been able to launch our application on the Microsoft HoloLens. Once the application was launched, we would have been able to perform a trace from the Windows Device Portal. After the trace finished, an ETL file would have been available to download. An ETL file can be opened directly in the Windows Performance Analyzer, which is a tool that would have allowed us to visualize the traces as graphs and tables that show the system and application performance. The following data could have been collected from the WPA software:

1. SoC power: System-on-chip power usage
2. System power: System power usage
3. Frame rate: Frames per second, missed VBlanks per second, and consecutive missed VBlanks
4. GPU: GPU engine usage, percent of total available
5. CPU: percent of total available
6. Memory: Total, in use, committed, paged, and non-paged

This information would have been used to compare the application running on both the headset and on the host computer. By doing this our team would be able to prove that by utilizing a host computer would indeed increase the battery life and performance of the HoloLens.

[Amy] Special Effects

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. The coin was created manually using this software and otherwise would not be available as a special effect. The coin design was used following an online tutorial because Blender use was unknown.

For a raining coin effect in Unity the created 3D graphic had to become an object with components that included a particle system which would allow the new 3D import to have moving capabilities such as rotation and dispersion. Providing a Rigidbody component In Unity allowed these 3D coins to have physics qualities like the effect of gravity on falling objects which was a desired effect for this graphic.

Testing the raining coins effect included exporting the FBX file into Unity. This file contained the lighting, 3D object dimensions, and the customized gold color created on Blender. Once the 3D graphic was in Unity there were many adjustments and settings that were changed to view the coin design in a new platform such as Unity. The color did not transfer well from software to software therefore a new custom gold color was created in Unity and added to the 3D graphic. The dimensions of the coin also did not automatically stranger over and adjustments in Unity allowed the coin to become 3D once in Unity.

To have more control of this new particle system in Unity the team wanted to include a script that would allow the particle system to just be called and seen when necessary. Having too many objects in every frame of the game takes up a lot of memory and can cause delays or even GPU crashes. Aside from that the team just wanted the coins to be shown whenever the player won the game. Without implementing the particle system to the other parts of the game to include the game logic to include this control aspect a switch was created using the keyboard as an input for turning the switch ON and OFF. This will change as testing progresses for this system.

Unity is a powerful and diverse platform that gives creators and designers the power to be creative by providing many resources to allow special effects to take place like sound and custom moves. Sound effects can be made possible as imports while customizing the moves of an object in the game can be made possible through a script file. For the raining coins to have a sound effect associated with them there was a WAV file uploaded with a jackpot sound from a casino. Attaching the file as an audio clip in an audio source, a component of the object, allowed this effect to take place.

The combination of these was not terrible to test because they worked as they were intended to. Once implementing them into the game logic is where sound failure was experienced. There was sound so it wasn't an absolute failure it just no longer sounded like hitting a jackpot. This caused the elimination of this sound effect from the winning special effects and only the raining of coins stayed. The losing sound effect stayed because it worked correctly.