HCI909 Project - Another Hi-Lo Game

by Artem Golovin and Nikolay Bezkhodarnov

Game rules

This game is based on the Hi-Lo game rules, where only ranks of cards matter. If you have the higher rank, then you win.

- The game requires 2 players and french playing cards (V for Jack, D for Queen, R for King, 1 for Ace)
- Each player has a deck of 52 cards (without Jokers), and the goal is to beat the opponent by playing cards with higher rank.
- The game lasts 15 rounds. At each round both players select 3 unique cards, specifying an order. Then, the rank of each card is compared to the opponent's corresponding card.
- The player with the higher rank gets a point. If the ranks are equal, no player gets a point. The player with more points at the end of a round wins it.
- The winner of the game is a player who won more rounds.
- You can use a unique card just once during the whole game match, so after the current round, cards become unavailable for selection in subsequent rounds

Advanced interaction techniques

- To play the game, you need to prepare a real (physical) french card deck, because the game is played with real cards. The card loading is controlled by a camera, which is used to detect, identify the cards and their order, so you need to choose a plain monochrome surface you are going to place your cards on (see Figure 1). Make sure you adjusted your camera well so that all cards fit into the picture and there are no problems with surrounding light, etc.
- The game is played with 2 players by a network connection. The players can be in different locations.
- The game can be controlled by user voice commands. The commands are used to control the game flow, but all the manipulations in menus preceding the game are controlled by mouse.

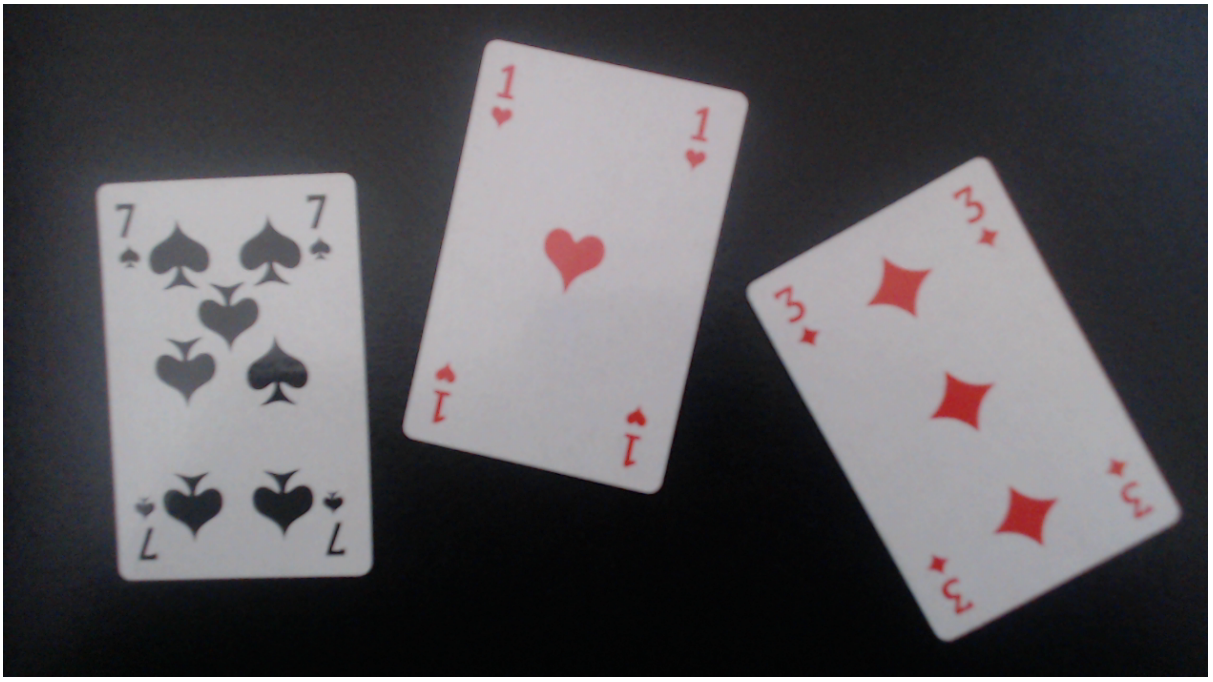

Figure 1. An example of how a surface and card layout should (could) look like

Installation

You need Python v3.11 (you could use the miniconda tool to conveniently manage Python environments and its packages).

Navigate to game folder and install requirements via running the next command:

pip install -r requirements.txt

After the last step everything should be ready to run the game.

How to run the game

Launch the game by running the following command in the project root directory:

python main.py

How to configure the game

After launching the game, go to settings (Figure 2) and choose the camera and the microphone you want to use (Figure 3). Then, you can start the game.

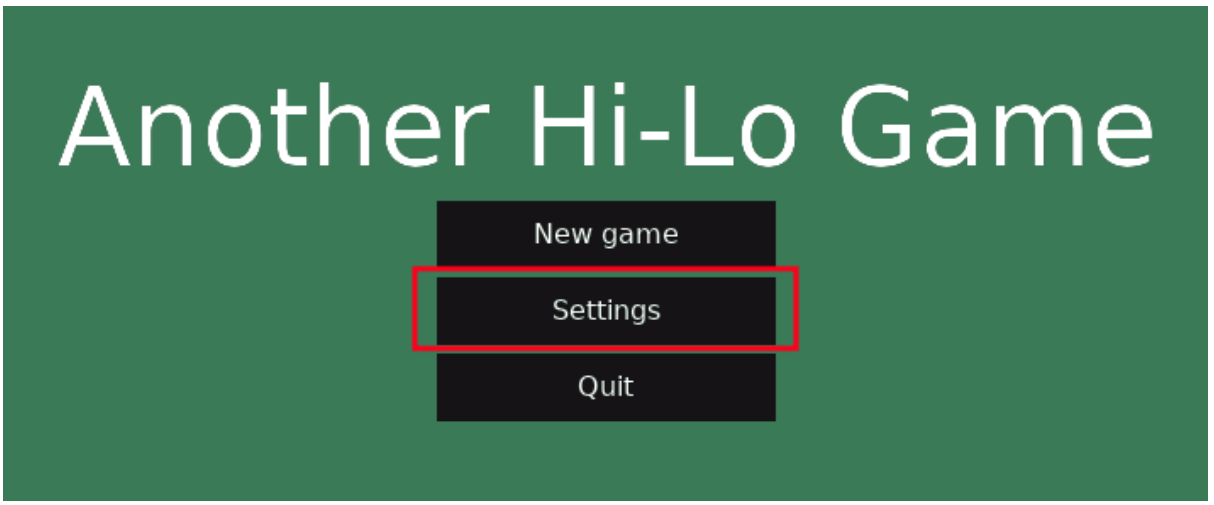

Figure 2. Main menu's settings submenu

Figure 3. Camera and Microphone selection menu

How to start the game

When you start a new game you can either create a new session hosting it or connect to a host by typing its' IP address in the host text field and pressing Connect button (Figure 4)

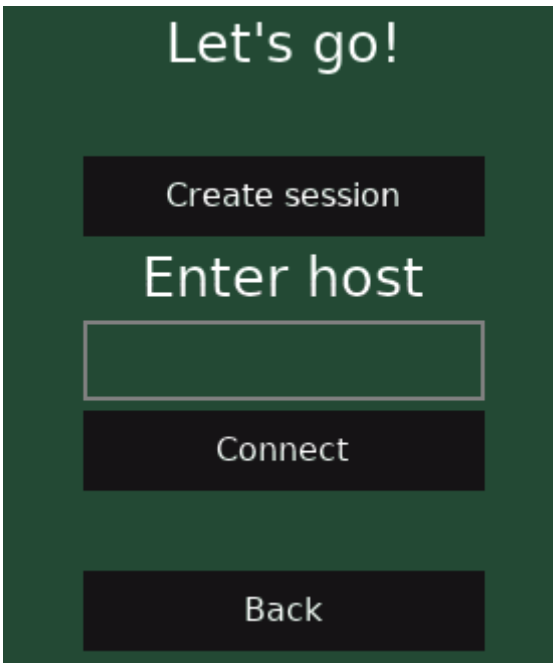

Figure 4. Starting game menu

Game flow

When you start the game you are going to see the next view:

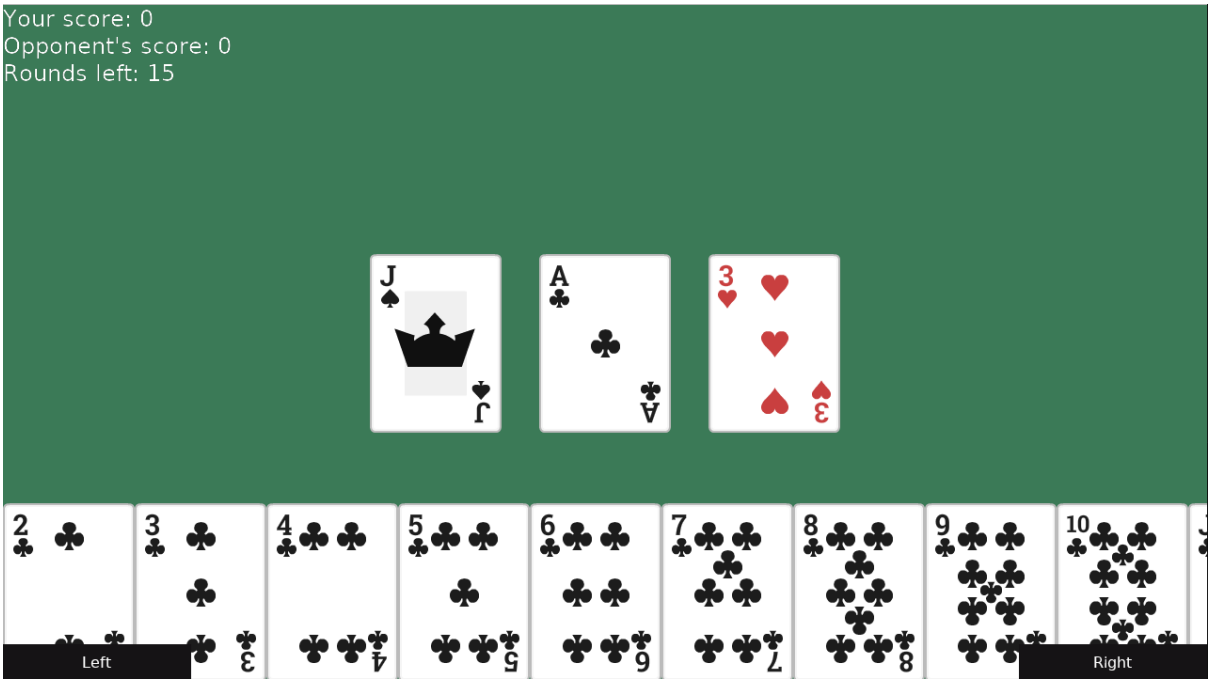

Figure 5. Main game view

In the top-left corner you can track the game statistics. **“Score”** means number of rounds you or your opponent won so far.

In the middle, identified cards from the camera stream are displayed.

In the bottom of the window available cards you can select from are displayed. You can scroll this list by pressing on **“Left”, “Right”** buttons.

There are a few steps you need to proceed to complete a round. The control over cards consists of several commands.

First of all, when you see that cards which are present in the middle of the screen are fine and they match the ones you placed on a surface you need to say **“Start”** (see Voice commands section). After this, you can either **“Switch”** 2 cards on the screen if you missed something (because the order of cards matters) or start to compete with your opponent by saying **“Ready”**.

When you are ready your cards are sent to the server, which processes it when all players are ready and provides you with the results afterwards.

When results are given, you will see the next screen:

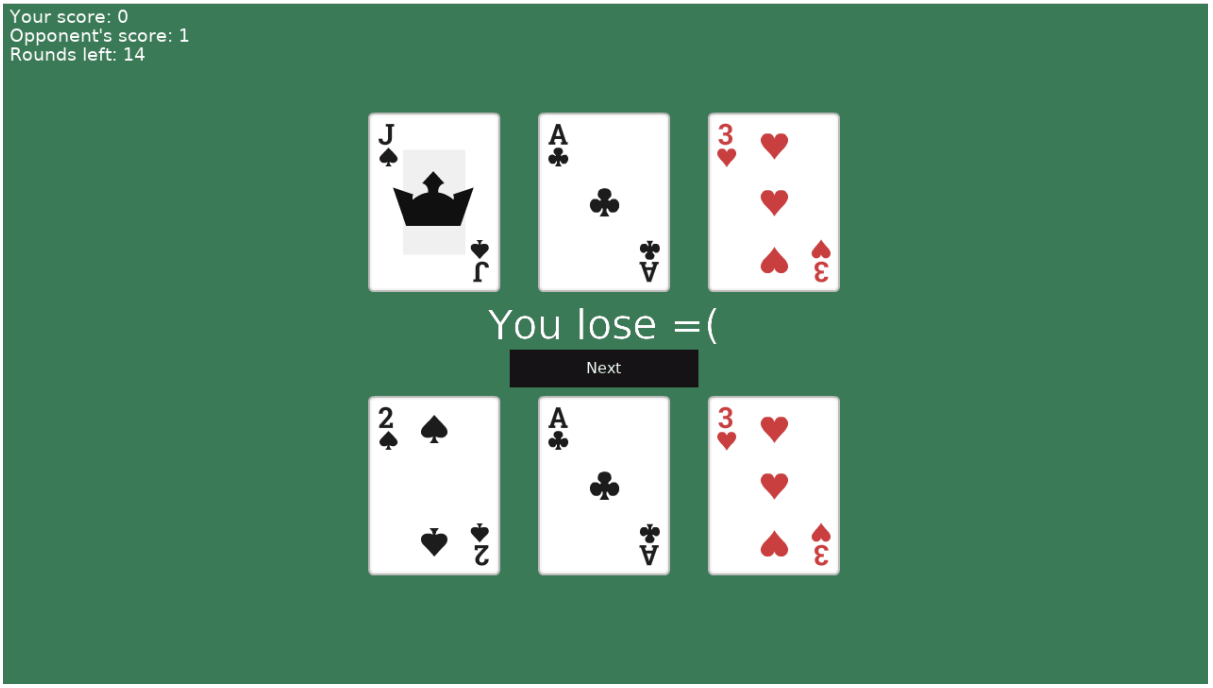

Figure 6. Round results view

Here, you should say **“Next”** to start another round or click on it.

When you have played all rounds the game result screen will be shown:

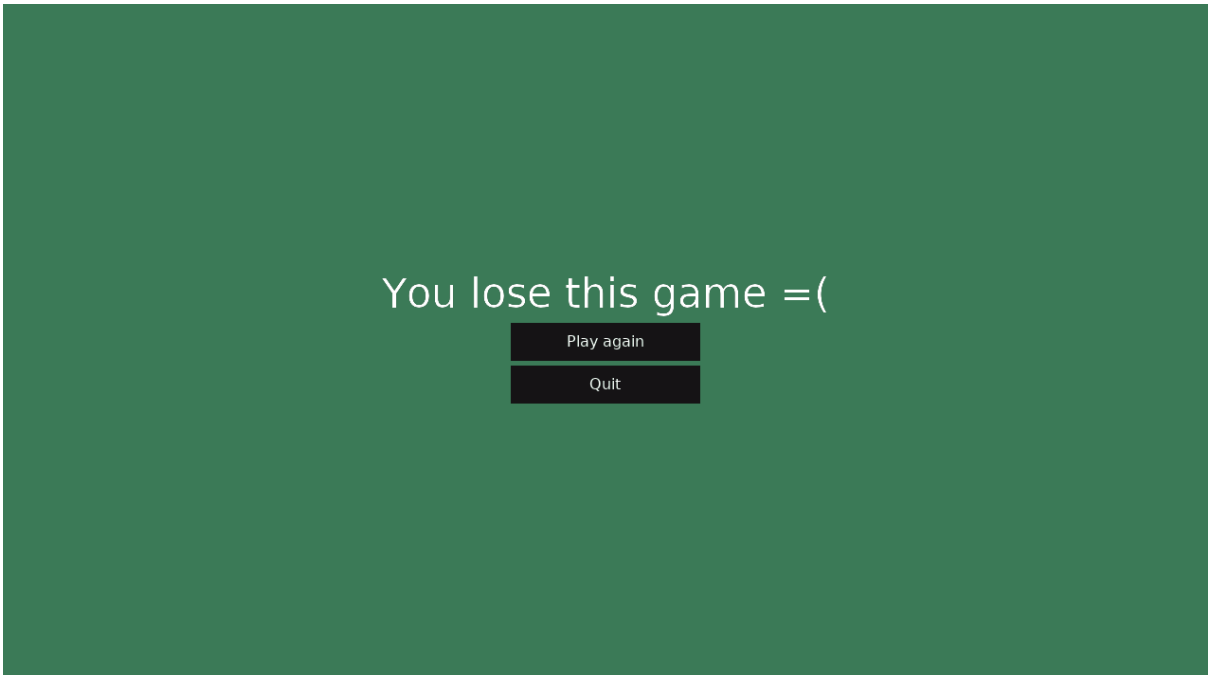

Figure 7. Game results view

Here, you can either start a new game by saying **“Play again”** or **“Exit”** or by clicking on it.

Voice commands

Available commands for a round:

- **Start** – Fixes cards from a camera. After that you cannot change them
- **Switch** – Switches any 2 cards on the screen that you mention with **Left**, **Middle**, **Right** keywords respectively. Example of a phrase: “switch left and middle”
- **Ready** – Sends cards to the server to compete with your opponent

Available commands for the Round results menu:

- **Next** – proceeds to next round after seeing the results

Available commands for the Game results menu:

- **Play Again** – Starts new game resetting player state
- **Exit** – Quits the game

Implementation specification

The game is implemented in Python 3.

The game uses the following libraries:

- SpeechRecognition (speech recognition)
- openai-whisper (speech recognition)
- pyaudio
- soundfile
- opencv-python>=4.8.1.78 (image processing)
- ultralytics==8.0.196 (CNN model for card detection)
- arcade==3.0.0-dev.25 (game graphics)
- janus (network communication supporting asyncio coroutines)
- v4l2py ; sys_platform == 'linux' (camera information)
- pygrabber ; sys_platform == 'win32' (camera information)

Python has been chosen to make this game, because it's a high-level language, which is easy to use and has a lot of libraries. Therefore, this allowed us to focus on game mechanics and improving the gaming experience, rather than spending a lot of time implementing basic functionality

The libraries have been chosen to make the game more interactive and to make it possible to control the game with voice commands.

For the arcade library we decided to use the development version (unreleased yet), because it has a lot of new features for building UI, which are not available in the stable version.

Implementation insides

To detect the cards from a camera, we use OpenCV capabilities and the YOLOv8 model.

The first step performed during the card detection of camera image (Figure 8) is to make a two-channel image in which the contours of the cards are clearly visible (Figure 9) using OpenCV. Next, we iterate over all found contours checking if it can be well approximated using 4 connected straight lines. If so, then most likely it’s a card.

Next step is to transform card contours to the “standard” position where ranks are pointing alongside the vertical axis (Figure 10). Then, we crop the top-left corner (Figure 11)

and use it as an input for the YOLO model. It allows us to be independent from a card deck you play with.

The model is trained on the dataset built from real french cards. The dataset is created by taking the upper left corner of the cards in different angles and in different lighting conditions (Figure 13). The dataset contains 52 classes, each class represents a card (suit and rank). The model is trained to classify them.

Important remark is that cards cannot overlap with each other and other objects (Figure 1), because it's a way harder task to solve, but a card can be placed at any angle to the camera, even placed horizontally.

After you use the corner image as an model input, if the confidence of prediction is less than 35% than we discard it.

Since the order of the cards matter we need to choose just 3 first cards if more cards are present which are sorted by the X axis which goes from left to right.

Figure 8. Origin image given as input to the card detector

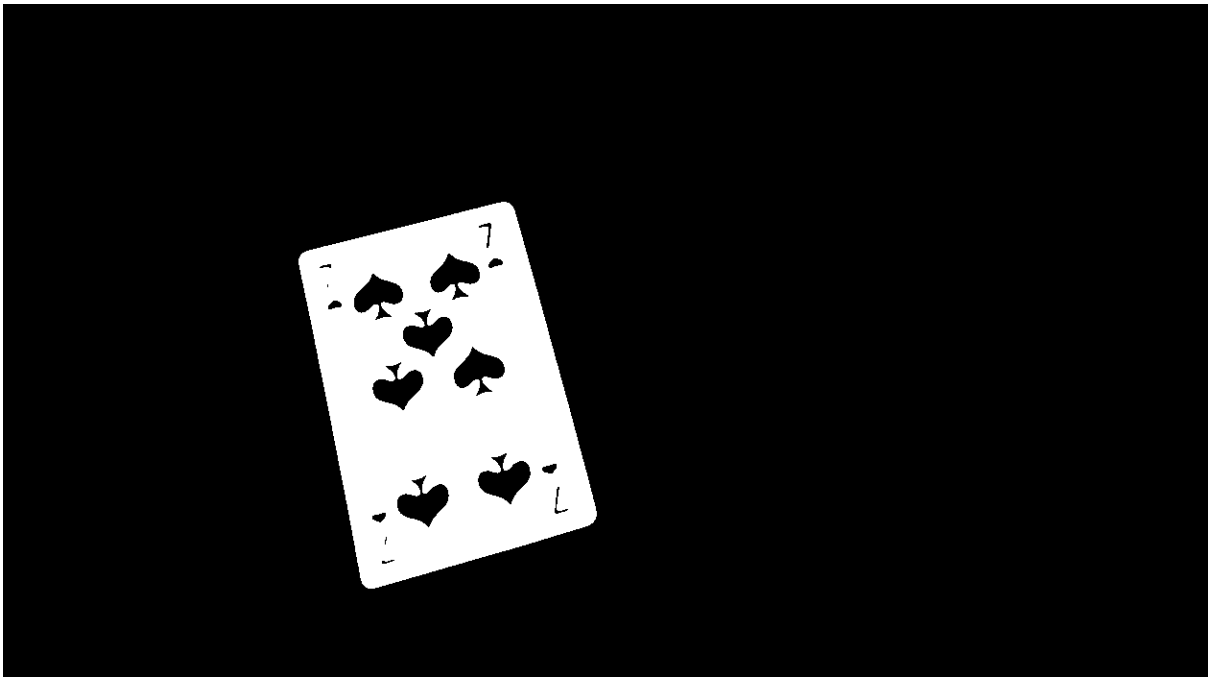

Figure 9. Thresholded image using OpenCV to find object contours on it

Figure 10. Card image transformed to “standard” position with the size of 320x448 pixels

Figure 11. Cropped top-left corner by 128x128 pixels

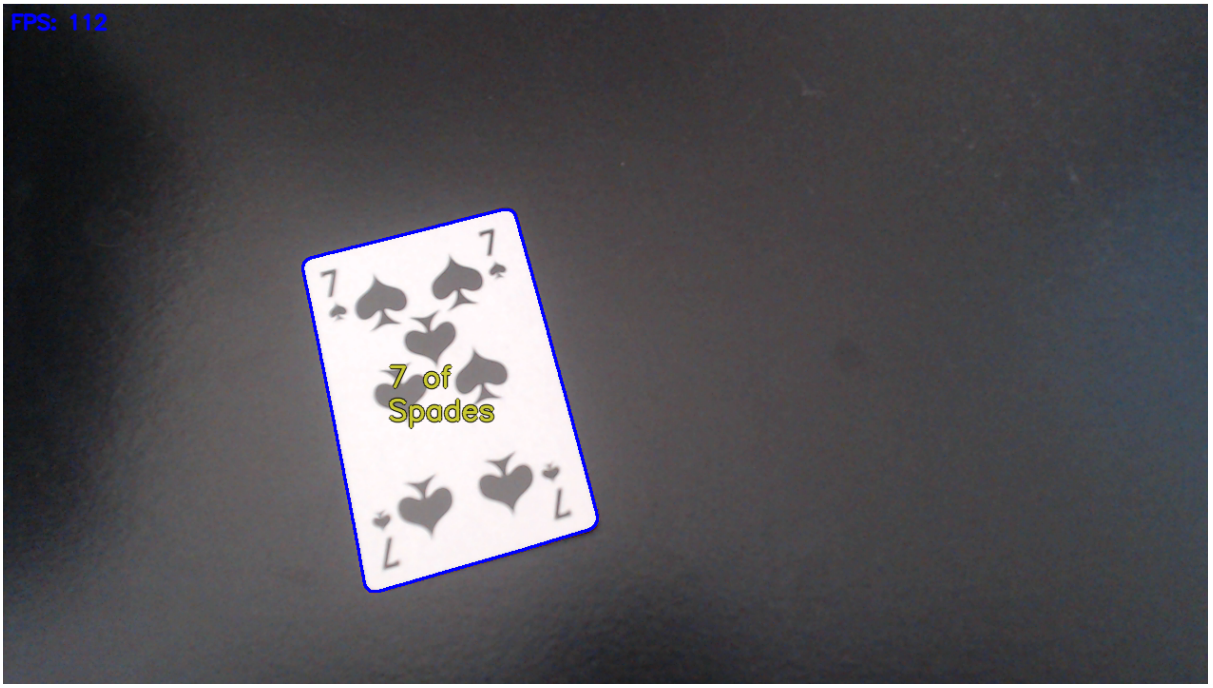

Figure 12. The results of card detection and identification shown on the origin image

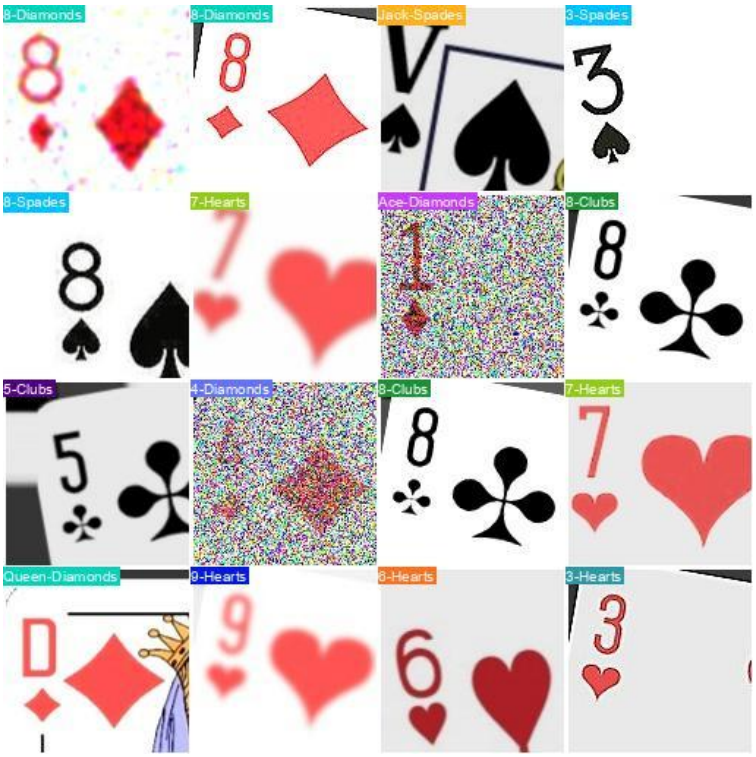

Figure 13. Augmented card corner images forming train dataset

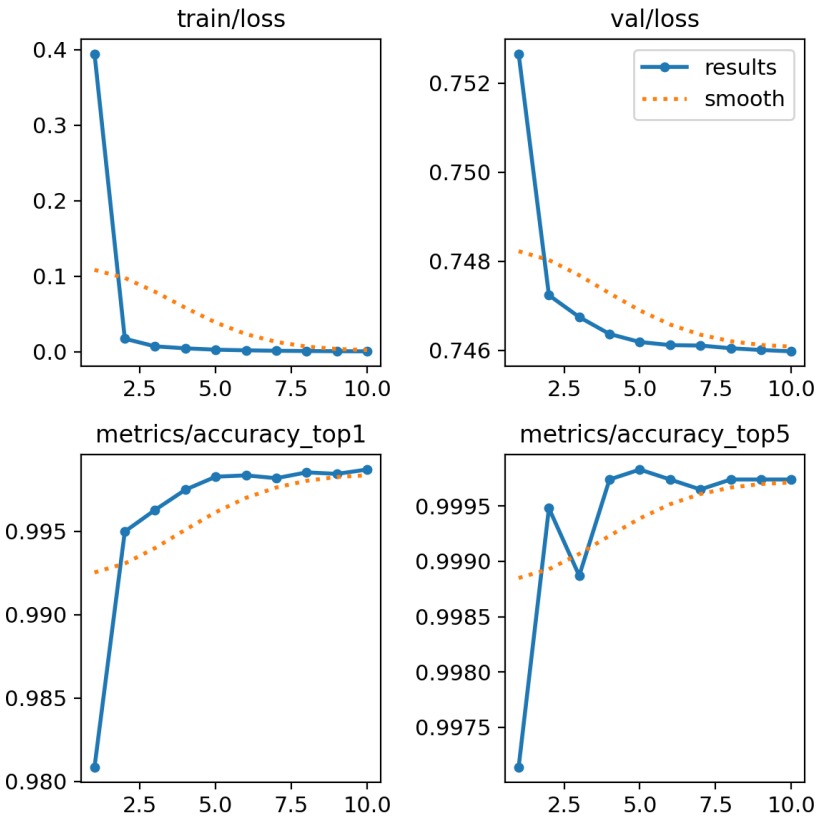

Figure 14. Training results

The YOLOv8 model was chosen by us because it has already fine-tuned models with ready-to-use architectures for specific tasks. All we need to do is just a couple of small tweaks and provide it with a dataset to train on. It was crucial for us since we are not ML guys and failed trying to build a model from scratch using pure tensorflow.

For the server side we decided to use Python Asyncio library because it was new for use, less wasteful than pure threads and well-suited to run small servers and it's in Python standard library, so well documented. We face a problem of separating Asyncio and Arcade libs since they are not supposed to run together in the same thread at least, so we decided at first to run the server as a subprocess but we realized that it's possible to configure it to run in the separate thread and makes it easier to debug.

To make the player's game process easier, we made an interface to show cards available to choose from. The problem was the arcade library (at least, in the stable version) doesn't provide any widget to show data that's bigger than the window. To solve this, we implemented the custom widget that places card sprites to the bottom of the window and two buttons: 'left' and 'right'. These buttons move all card sprites in a corresponding direction. In this way it becomes possible to observe available cards even in a situation, when the window size isn't big enough to show all of them.

To recognize voice commands, we use the SpeechRecognition library, which supports many voice models. We decided to use one that is developed by the OpenAI company called Whisper, because it works offline and has the highest accuracy of all tested offline models. For the online models you need to either have a paid subscription or there is

a daily limit of phrases it can recognize for you. However, it should be noted that this is the part of our project that we can least influence, so we cannot improve the quality of the models and unfortunately not all of them work well.

Also, for debug purposes we have a constant called DEBUG_SESSION which is just a boolean variable indicating if we want to have debug things like extra buttons, some extra logging, etc. By default it's set to True.

Also, if you work in Linux system most likely you will see the next warnings which are common for most Linux users of Speechrecognition library:

ALSA lib pcm_dsnoop.c:601:(snd_pcm_dsnoop_open) unable to open slave

ALSA lib pcm_dmix.c:1032:(snd_pcm_dmix_open) unable to open slave

ALSA lib pcm.c:2664:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.rear

ALSA lib pcm.c:2664:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.center_lfe

ALSA lib pcm.c:2664:(snd_pcm_open_noupdate) Unknown PCM cards.pcm.side

ALSA lib pcm_oss.c:397:(_snd_pcm_oss_open) Cannot open device /dev/dsp

ALSA lib pcm_oss.c:397:(_snd_pcm_oss_open) Cannot open device /dev/dsp

ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card

ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'

ALSA lib confmisc.c:160:(snd_config_get_card) Invalid field card

ALSA lib pcm_usb_stream.c:482:(_snd_pcm_usb_stream_open) Invalid card 'card'

ALSA lib pcm_dmix.c:1032:(snd_pcm_dmix_open) unable to open slave

Don't worry, it's okay and still works.

Conclusion

In conclusion, we would like to say that we've learned some new things while we were doing it. There are ways to improve this project, for example change the CNN model to a one which can handle card overlapping, make the voice control more stable, etc. But so far we are glad with the results.