Supplemental materials for midterm presentation
Group 6 - Hieu Nguyen

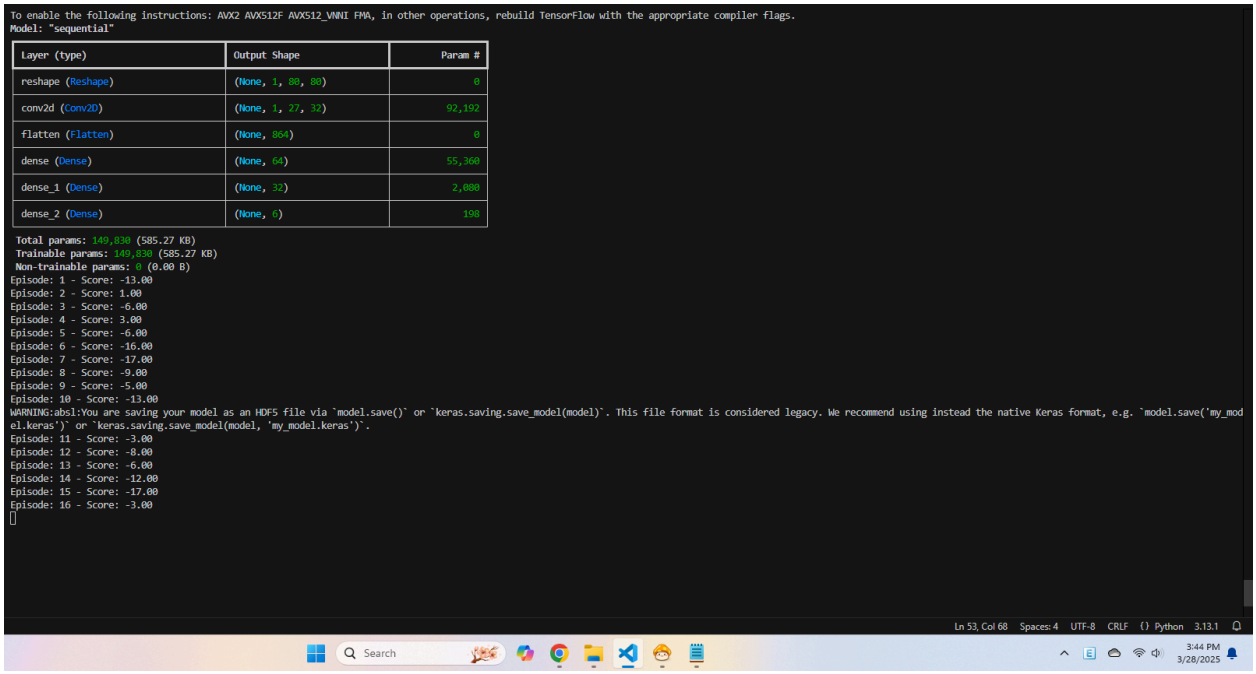
- Our python code is working after I change the library from gym (which has been deprecated) into Gymnasium (v1.1.1) and change some function variable name to match the newer version of Tensorflow too because the format of the model from the repository I believe was developed using Tensorflow v2.13 or older that still use the .h for saved trained model. Instead we use Tensorflow v2.16 which support .keras model. There are several old and new version compatible problem but by changing the code slightly we are able to get it working.
- For example in the architecture of the network we use different declaration

```
def _build_model(self):
    model = Sequential()
    model.add(Reshape((1, 80, 80), input_shape=(self.state_size,)))
    model.add(Convolution2D(32, 6, 6, subsample=(3, 3), border_mode='same',
                            activation='relu', init='he_uniform'))
    model.add(Flatten())
    model.add(Dense(64, activation='relu', init='he_uniform'))
    model.add(Dense(32, activation='relu', init='he_uniform'))
    model.add(Dense(self.action_size, activation='softmax'))
    opt = Adam(lr=self.learning_rate)
    # See note regarding crossentropy in cartpole_reinforce.py
    model.compile(loss='categorical_crossentropy', optimizer=opt)
    return model
```

Changed to

```
def _build_model(self):
    model = Sequential()
    model.add(Reshape((1, 80, 80), input_shape=(self.state_size,)))
    model.add(Conv2D(32, kernel_size=(6, 6), strides=(3, 3), padding='same',
                     activation='relu', kernel_initializer='he_uniform'))
    model.add(Flatten())
    model.add(Dense(64, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(32, activation='relu', kernel_initializer='he_uniform'))
    model.add(Dense(self.action_size, activation='softmax'))
    opt = Adam(learning_rate=self.learning_rate)
    model.compile(loss='categorical_crossentropy', optimizer=opt)
    return model
```

- Working code evidence:
  This is the terminal when running the code to train without enabling the interface for faster execution.



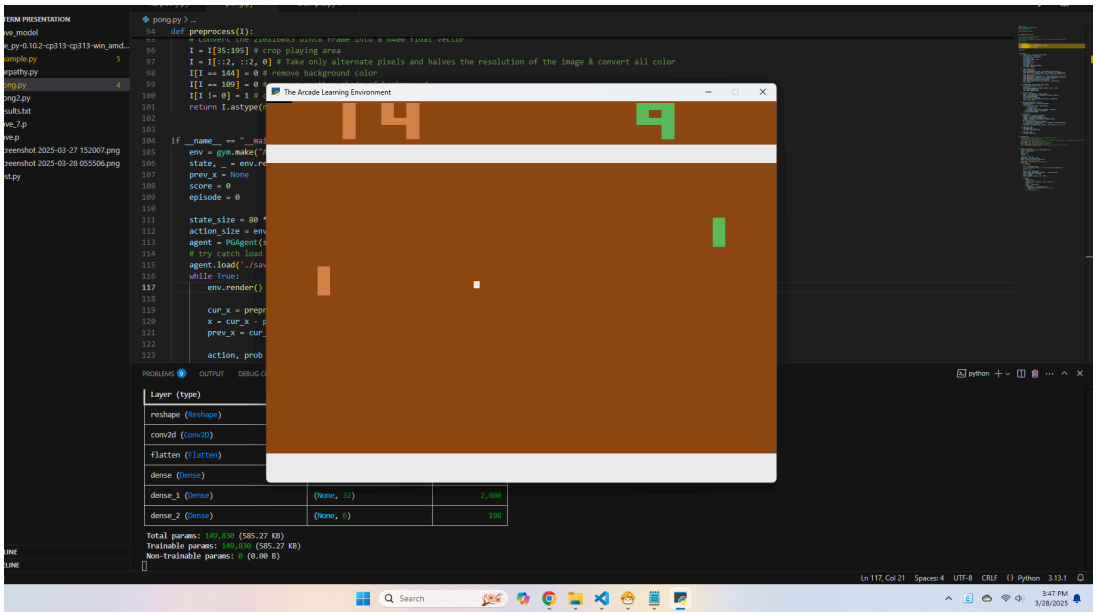
```
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Model: "sequential"
```

Layer (type)	Output Shape	Param #
reshape (Reshape)	(None, 1, 80, 80)	0
conv2d (Conv2D)	(None, 1, 27, 32)	92,192
flatten (Flatten)	(None, 864)	0
dense (Dense)	(None, 64)	55,360
dense_1 (Dense)	(None, 32)	2,080
dense_2 (Dense)	(None, 6)	198

```
Total params: 149,830 (585.27 KB)
Trainable params: 149,830 (585.27 KB)
Non-trainable params: 0 (0.00 B)
Episode: 1 - Score: -13.00
Episode: 2 - Score: 1.00
Episode: 3 - Score: -6.00
Episode: 4 - Score: 3.00
Episode: 5 - Score: -6.00
Episode: 6 - Score: -16.00
Episode: 7 - Score: -17.00
Episode: 8 - Score: -9.00
Episode: 9 - Score: -5.00
Episode: 10 - Score: -13.00
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
Episode: 11 - Score: -3.00
Episode: 12 - Score: -8.00
Episode: 13 - Score: -6.00
Episode: 14 - Score: -12.00
Episode: 15 - Score: -17.00
Episode: 16 - Score: -3.00
```

By setting render_mode to “human” the environment will render the visualization of the state.

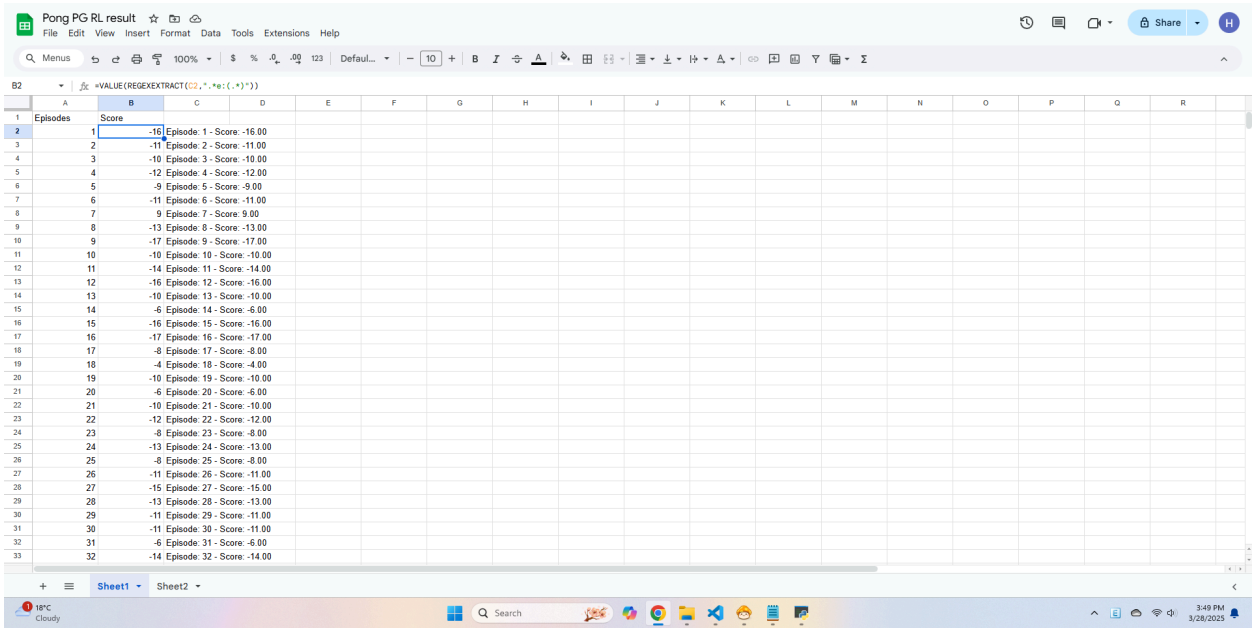
```
env = gym.make("ALE/Pong-v5", render_mode="human")
```



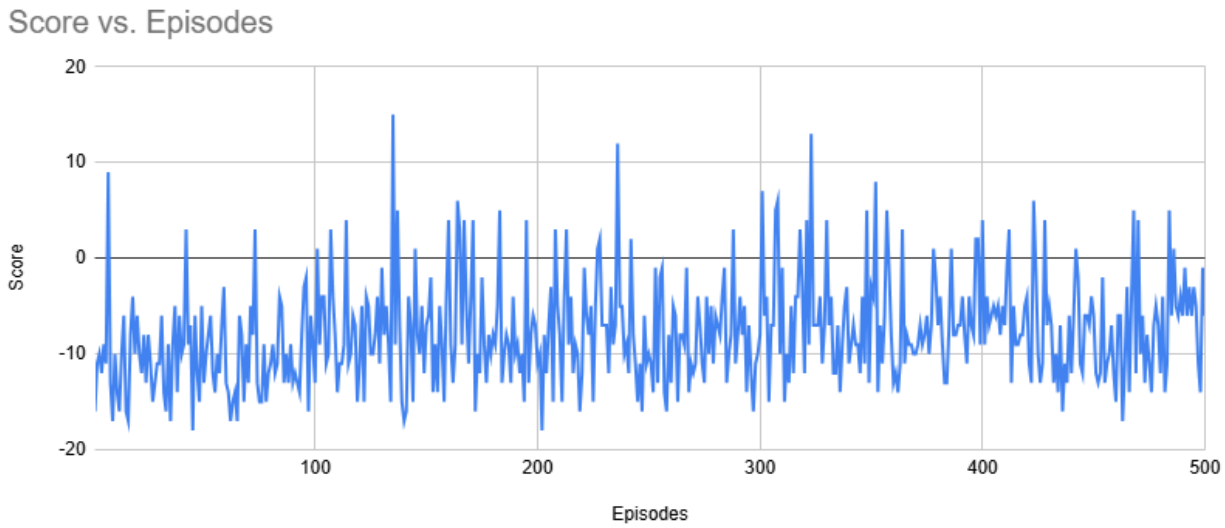
In the original version each episode will be play until someone reach 21

points and then subtract the score to get the final score of that episode like in the first picture.

- Since we can not re use the trained model on the repository the author upload, we have to train it from the beginning and after 500-1000 episodes the model slightly improve but still no where near beating the computer. We extracted the result into sheet and create a plot with the result like we show in our presentation.



Episodes	Score	
1	-16	Episode: 1 - Score: -16.00
2	-11	Episode: 2 - Score: -11.00
3	-10	Episode: 3 - Score: -10.00
4	-12	Episode: 4 - Score: -12.00
5	-9	Episode: 5 - Score: -9.00
6	-11	Episode: 6 - Score: -11.00
7	9	Episode: 7 - Score: 9.00
8	-13	Episode: 8 - Score: -13.00
9	-17	Episode: 9 - Score: -17.00
10	-10	Episode: 10 - Score: -10.00
11	-14	Episode: 11 - Score: -14.00
12	-16	Episode: 12 - Score: -16.00
13	-10	Episode: 13 - Score: -10.00
14	-6	Episode: 14 - Score: -6.00
15	-16	Episode: 15 - Score: -16.00
16	-17	Episode: 16 - Score: -17.00
17	-8	Episode: 17 - Score: -8.00
18	-4	Episode: 18 - Score: -4.00
19	-10	Episode: 19 - Score: -10.00
20	-6	Episode: 20 - Score: -6.00
21	-10	Episode: 21 - Score: -10.00
22	-12	Episode: 22 - Score: -12.00
23	-8	Episode: 23 - Score: -8.00
24	-13	Episode: 24 - Score: -13.00
25	-8	Episode: 25 - Score: -8.00
26	-11	Episode: 26 - Score: -11.00
27	-15	Episode: 27 - Score: -15.00
28	-13	Episode: 28 - Score: -13.00
29	-11	Episode: 29 - Score: -11.00
30	-11	Episode: 30 - Score: -11.00
31	-6	Episode: 31 - Score: -6.00
32	-14	Episode: 32 - Score: -14.00



- The analysis after watch the model play the game is that the model did improve on getting score by 1 bounce only but if the computer can

catch the ball and bounce it back again the model don't know how to react yet. Because the model can only score after 1 bounce is not enough to win the whole set of the game which will most of the time contain multiple bounces back and forth therefore the result is still low. On the brightside the model did learned how to score with 1 bounce resulted in some lucky match win against the computer hence the high spike in the chart. We believe that with more training episodes the model will started to learn and beat the computer at some point.

- So after some few research studies we found another solution for training Pong using reinforcement learning by professor Karpathy using slightly different method but same preprocess and game loop. We also have to update the code to match our current version of Tensorflow to get it to work.
- Here are the architecture comparison table of 2 methods

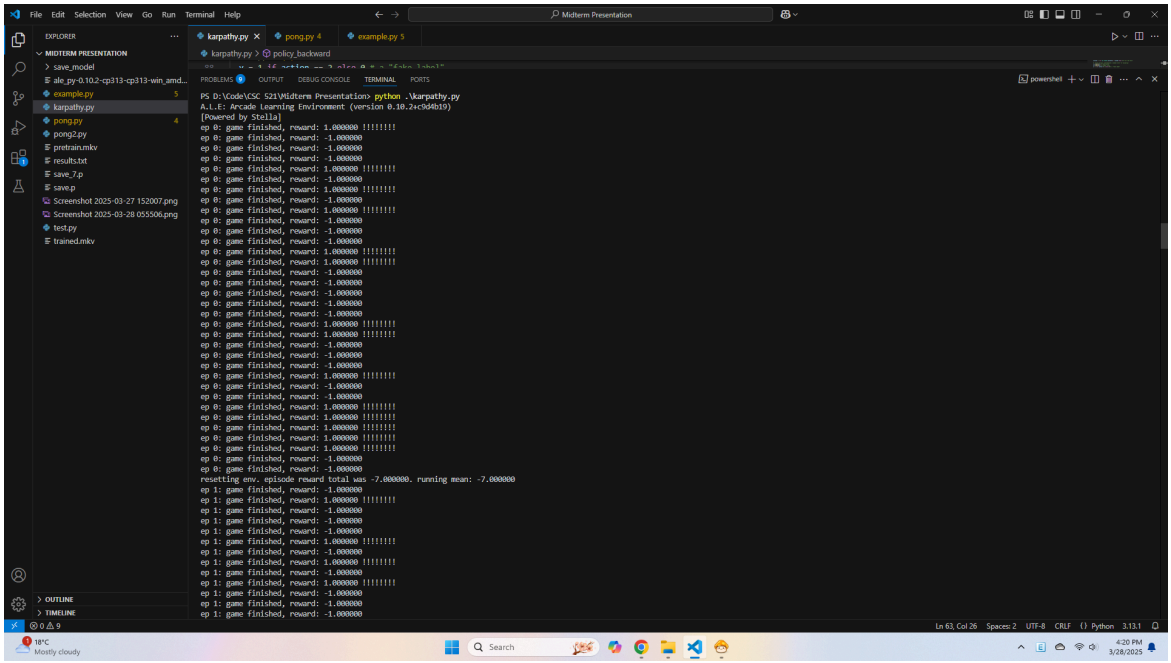
Feature	PGAgent - Keras	Karpathy's
Neural Network Type	Convolutional Neural Network (CNN)	Fully Connected (2-layer)
Input Preprocessing	Converts to grayscale, downsamples, and flattens	Similar preprocessing
Hidden Layers	Conv2D → Dense (64) → Dense (32)	200 ReLU neurons
Output Activation	Softmax (multi-action policy)	Sigmoid (binary action)
Action Selection	argmax on probabilities	Probability-based (binary decision)
Gradient Update	Uses categorical_crossentropy loss	Custom backprop with manual weight updates

- And the performance comparison

Training Speed	Slower due to CNN	Faster (fewer computations)
Memory Usage	Higher (stores entire CNN model)	Lower (stores only two weight matrices)
Action Space	Uses softmax over all actions	Chooses between two actions (2 or 3)

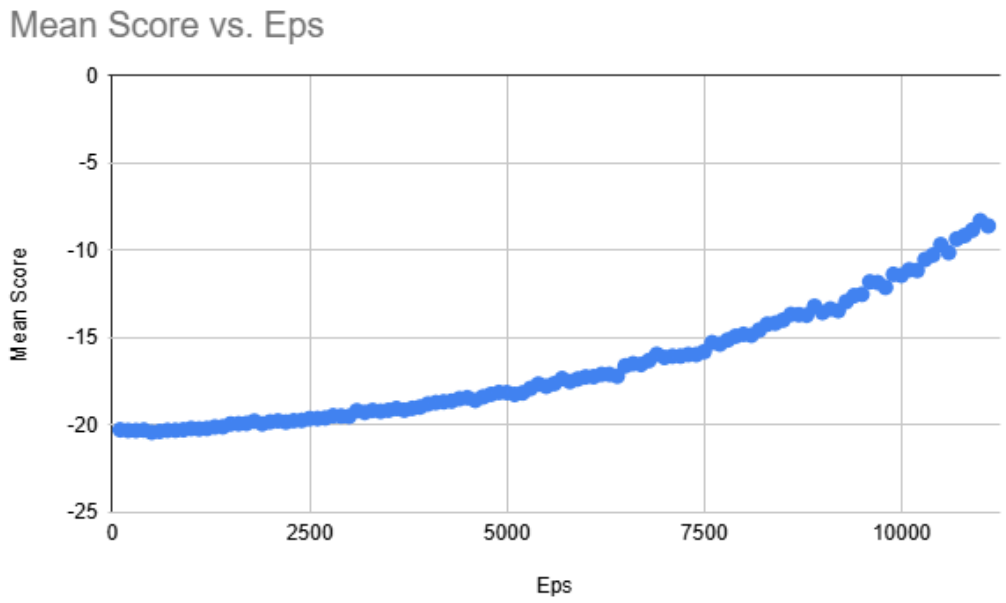
Observation: Like I mention in the presentation the environment for Pong by Gymnasium contain 6 possible actions but in reality only 2 action need to be used (up & down) and the code by professor Karpathy only focus on these 2 actions and set the probability of other 4 into 0 from the beginning I think made a great optimization for the model learning speed.

- Conclusion:
    - Karpathy's code is faster since it doesn't use deep learning and avoids TensorFlow overhead.
    - PGAgent can potentially learn better representations due to CNN features.
- So after running the code by professor Karpathy we have been able to get meaningful result.

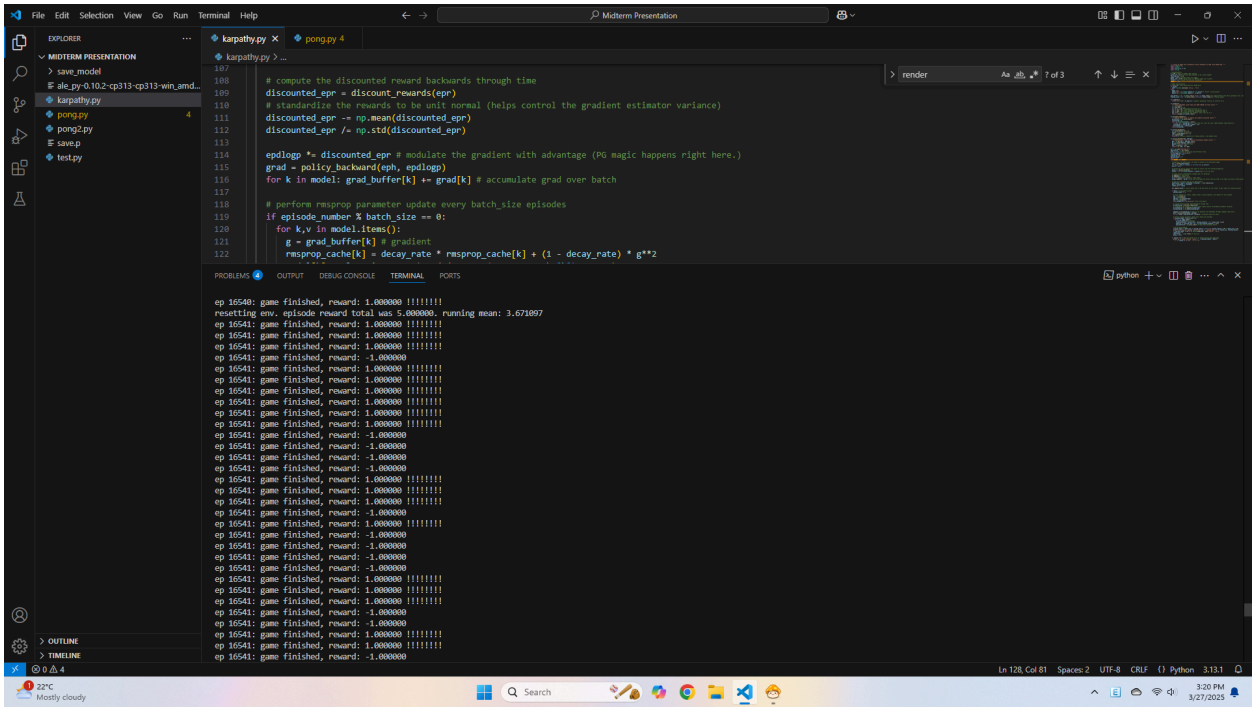


The code output all set in 1 game of Pong and count each set win or lose

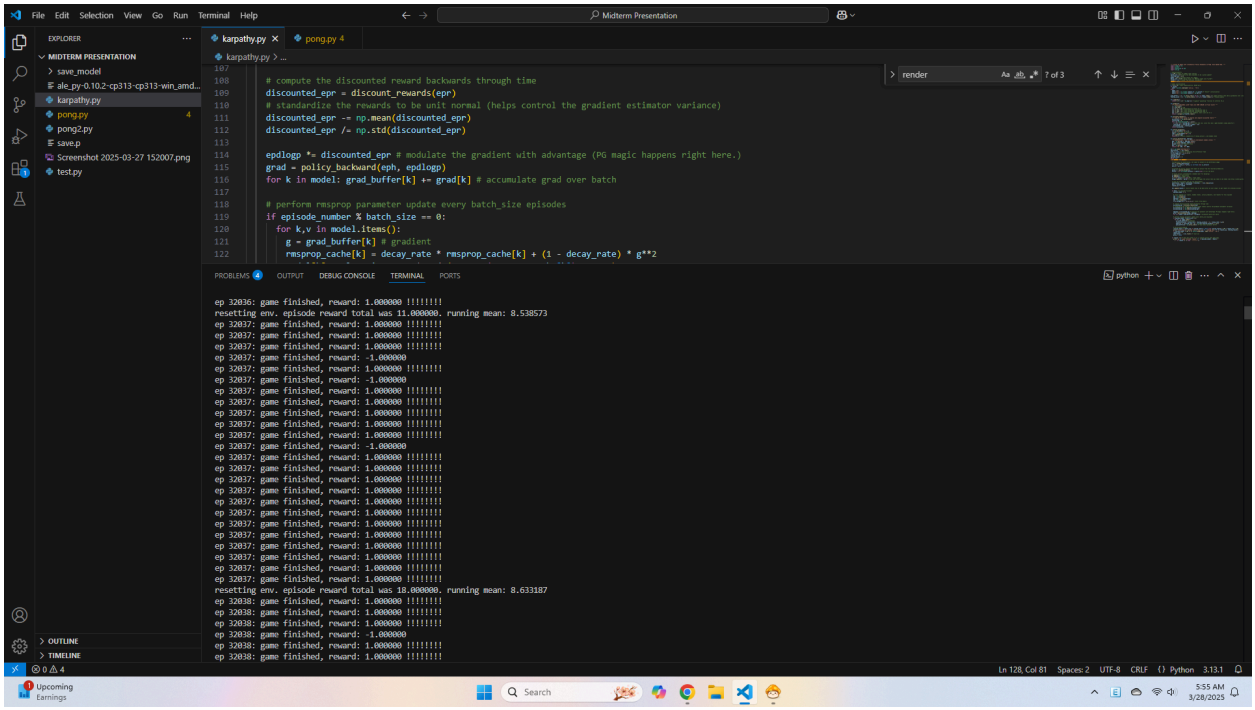
is -1 or 1 points and calculate the mean points after each episodes. At the start of the training there were no model so the result is frequently -21 because the newly created model couldn't score a point.



But after training for about 2500 episode the modal start improving and by 12000 episode the mean score was -7. I did train the model a little bit more to get the mean score to be 3.6 by around 20000 episodes



And 8.5 mean score by 42000 episodes but unfortunately I did not implement a result saving function for it so I can only add 2 snapshots of the running code with align time stamp for evidence.



The result get for the chart above is when I retrain the model from the beginning and try to map it as a chart before the deadline so it is not complete but from both the chart and the snapshot we can see that after 42000+ episode the model we trained started to win a lot more than the computer. I also record a short video of a pretrain and trained model playing the game it self and will attach it in the zip file.

- Finally thank you professor for introduce us to an interesting subject and please let us know if you need any more proofs.

Result links:
https://docs.google.com/spreadsheets/d/1BS7hY0kv2BqskTP3oVNnSH1oamYaD7HVzzVMHeOd_zw/edit?usp=sharing