Report Documentation

Jeroen Arts

Index

[1. Preface 3](#_Toc124413045)

[2. Data Preparation 4](#_Toc124413046)

[2.1 breakout 4](#_Toc124413047)

[2.2 Pong 5](#_Toc124413048)

[2.3 Blackjack 5](#_Toc124413049)

[2.4 Taxi 5](#_Toc124413050)

[2.5 Cart Pole 6](#_Toc124413051)

[2.6 My data 7](#_Toc124413052)

[3. Machine Learning 9](#_Toc124413053)

[4. Data Visualization 11](#_Toc124413054)

[4.1 Breakout Visualization 11](#_Toc124413055)

[4.2 Pong Visualization 13](#_Toc124413056)

[4.3 Blackjack visualization 15](#_Toc124413057)

[4.4 Taxi Visualization 17](#_Toc124413058)

[4.5 Pole keeper Visualization 19](#_Toc124413059)

[4.6 Conclusion 20](#_Toc124413060)

[5. Closing words 21](#_Toc124413061)

# Preface

Based on the work that I have done for my personal project and the feedback I got from the coaches, I decided to make a document in which my progress when working on the personal challenge is recorded. Also the reasoning behind some of my choice will be displayed and the conclusions I got from the challenge. This document is thus meant to report what I have done. It’s not my intention to change ones mind about reinforcement learning agents or to give a professional opinion on which agent is best. I shall only be using the knowledge I got from doing my challenge and everything written down here.

My original personal project was the making of an AI that could play the ‘Nine Men’s Morris’ board game. During the course of my project, this goal changed from making an AI that could play a single game, to seeing the difference between fresh AI’s playing multiple different games. For that end, I ended up using three different AI’s in 5 different game environments.

# Data Preparation

Starting with the data, it’s obvious that I didn’t use any open source data sets or any data sets at all, because my challenge uses Reinforcement Learning agents. All my data, which means the data used by the agents to learn, comes from the environment. Since multiple environments are used, multiple sets of data will be used for the different agents. However that is not the only data that is used by me. The training logs from the agents shall be used to compare them, the end result doesn’t interest me. What I want to see is the way the AI took to get there and thus their learning process.

## breakout

The first set of data comes from the Atari game breakout. Two different agents trained on it, those being the A2C agent and the DQN agent. As with most Atari game, the RGB image that players see will be returned to the agents when they are playing or training. This is in the form of a n-dimensional continuous space in which each pixel is written. Outside of the environmental data, the rewards the agent gets are visible to it. The agent observers the space and makes actions based on it.  
Afbeelding met tekst, schermafbeelding

Automatisch gegenereerde beschrijving

Figure 1 Atari Breakout Image

Other then the positions of the pixels it’s also possible to sent the state of the 128 bytes of Ram from the console to the agent or to sent a greyscale picture instead. The RGB image is the default observation environment that is used by the agent. As the only difference between the learning process of the greyscale and colour is the way the agent learns that higher place blocks have higher values. The RGB image is the default observation and using the right policy (CnnPolicy for images) is more then enough for each agent to train.

## Pong

The second data set comes from the Atari game Pong. The observations that the agent makes and thus the data that it gets from the Pong game is the same as with the Breakout game. Either a RGB/Greyscale picture or the state of the 128 bytes of Ram from the console. Just like with Breakout, I made use of the RGB picture. The agents can see the location of every pixel in a n-dimensional continuous space. The reasoning for using the RGB picture instead of the Console RAM or a Greyscale is that it is the default and fits my purposes for training the agent.

Afbeelding met tekst, groen, apparaat, meter

Automatisch gegenereerde beschrijving

Figure 2 Atari Pong game

## Blackjack

Blackjack is something entirely different from Atari games and thus the environmental data the agents will get is also different. From the environment the agent will get a tuble variable describing the current sum of the players cards, the value of the dealers card and whether the player is holding an ace or not. Since Blackjack is a mathematical game there is no need for more data from the environment. With this environment, I don’t have any choice or say in the way that data is delivered to the agent, since I can’t change the way the environment works underwater.

## Taxi

The fourth environment being use is the Taxi Game where the AI needs to get the taxi from one of the coloured spots towards the spot with the passenger and then back towards the destination. The observable space for this environment is a discrete variable holding the 500 different states that the taxi can be in. Each state can be described as a tuple holding the values: (taxi\_row, taxi\_col, passenger\_location, destination) describing in which row and column the taxi is and where the passenger location and destination are. The tuple gets encoded and saved as a discrete value. Encoding and Decoding are done underwater and are not visible. Just like with the Blackjack environment, there is no way for me the change the way that the agent observes the data.

Afbeelding met tekst, elektronica

Automatisch gegenereerde beschrijving

Figure 3 Taxi game

## Cart Pole

Pole, unlike the other environments, isn’t a game environment. It instead falls under the control category. However it can be seen the same as a game where you have two action which you must use to keep the pole upright for as long as possible. The observable space in this environment is an ndarray. In that array the cart’s position and velocity are displayed along with the Pole angle and angular velocity. Base on as the array gets updated the agents can ‘see’ what happened as result of their action or inaction.

Afbeelding met tekst, scorebord, getoond, schermafbeelding

Automatisch gegenereerde beschrijving

Figure 4 Cart Pole ndarray table

## My data

When training my agent a standard log will be generated describing some values of the training. This log shows me the data about the path the agent took to get where it got. This data will be shown in Data Visualisation, but an image of one of the logs is shown below. The reason I will mostly use the episode length and the rewards is because this is something that all the training logs of the agents show. Not all agents show the Learning rate or the amount of N-updates and thus it doesn’t become possible to use that data. There is also the fact that the logger is coded into the source data of Stable Baselines which I use for my agents and so I can’t see or change anything about it.

Afbeelding met tekst, scorebord, naamplaatje

Automatisch gegenereerde beschrijving

Figure 5 Training logs

This log shows the data that I get from a training session. What I will use from this is mostly the ep\_len\_mean and the ep\_rew\_mean. This is the average episode length and the average reward taken over the average of the logging interval. In the chapter Visualisation I will show the graphs from the entire training sessions.

Each environment gives between 20 and 150 logs depending on the agent and the environment. In the command for the agent to learn there is a variable that describes how big the interval is between logs being posted. For most environments I set that interval to 250 iterations. However for the DQN-agent in the blackjack environment I had to put it to 1 iteration between logs. Since each episode in the blackjack environment takes more timesteps then in the breakout environment, I would get less episodes done in the same amount of timesteps then with other environment and as such the interval must become smaller in order for me to get more, if any, logs. On the other hand, I could train the agent for more timesteps in order to get more logs, but that would drastically increase the training time so lowering the interval will give me more data in less time. After viewing all the logs I did notice that the DQN-agent gives less logs then the A2C-agent does. To still get enough data I put the interval from 250 like with the A2C-agent to 10. This gives me about the same amount of logs from the DQN-agent as the A2C-agent gives.

A big problem I came across was that while the PPO-agent did learn and did train, there were no logs and so I couldn’t see what it was doing. However the same solution that made it so that the DQN-agent gave more logs also let me use the PPO-agent. By setting the interval between logs to 1 I finally managed to get training logs from it. This increases the amount of data that I can use and show by 50%. This did mean I would only get around 20 logs from the PPO-agent depending on the environment.

# Machine Learning

For my personal project I, originally, wanted to make an AI that could play the game ‘Nine Men’s Morris’. However I had no prior experience with Python or with AI’s. In order to learn how to code in Python and how to use an AI, I followed a tutorial. In this tutorial we used Stable Baselines and OpenAI Gym in order to train AI that can play the Atari game Breakout. The tutorial then showed how to train an AI for an unnamed racing game. With that knowledge I decided to try and make my Morris AI. However I did not succeed. Because of the tutorial, I had some experience with the Gym environments, so I wanted to make the game board in Gym. Making the environment was harder then it looked, so in order to get more familiar with it I tried an easier game first.

I managed to find an environment for the board game ‘Reversi’. Using this as a base I trained some agents to get a feeling for how to combine the environment with the agents. After that I was planning on changing the Reversi environment until it became Mills. However while training the agents in Reversi and viewing the results, I found some interesting things. The A2C agent, which I used first, seems to stop learning after a while. If I tried with a fresh agent later then the end result was the same. It just kept repeating the same set of moves again and again. Resulting in a game that was always the same as the opponent did the same moves in order to get the easy victory. At that point I was curious and put a different type of agent (DQN) on the game. The results was vastly different. This agent took longer to train, before showing results and did keep learning for longer. Within the training time, it didn’t look like it stopped learning and when I evaluated the DQN-agent, it did try different moves during separate games.

My original goal of making an AI that can play the board game changed. I wanted to find out how the different agents played the same game. Stable Baselines gives me access to 7 different agents. I wrote the code for the 7 agents to play but in the end only 3 agents could actually play the game. This was because the agents have set spaces in which they can take actions. Most of the agents can only take actions in a Box space, while my game was a Discrete space. This left me with 3 agents of which 1 (PPO) did show any logs when I trained it. This has been solved far later into the minor.

In order to satisfy my curiosity I decided to have the two agents (A2C&DQN) that work play many different games instead of just 1 game. So I wrote the code for the agents to play: Breakout, Pong and Taxi. I let the agents train and reviewed the logs that came from the training. However I was satisfied with the training. When it came to Breakout and Pong it took a long time for the agents to finish and my CPU was for 100% active. This in turn meant that my CPU got very hot (85-90°C), I didn’t like that and looked into some ways to lower the load on my CPU and thus lower the temperature on it. I also wanted to speed up the training time. Considering that I have a proper GPU in my laptop, using it instead or in combination with my CPU became the obvious answer. However having the agents use my GPU wasn’t as easy. Stable Baselines is compatible with Nvidea Cuda, which is toolbox that can be used withcertain GPU’s to process data alongside the CPU. I went and checked that my GPU is compatible and then downloaded the software for it. However this was not enough. For some reason the code didn’t want to enable Cuda when training the agents. It took me a few hours and a dive into the documentation of Stable Baselines to find out how to force the agents to use Cuda instead of just my CPU. But that also wasn’t enough, at that point I tried a few different things and I’m not sure when I did it, but I eventually succeeded. Myy agents were using Cuda to train after I managed to get Cuda to work. The training time for my entire program went from 14 hours to just around 8 hours, while keeping the load on my CPU at around 70-80% and the temperature at around 60-70°C.

Since the purpose of my personal project became the logging of training data from the agents to learn the difference I decided to increase the amount of environments in order to get more data. I added Blackjack and Pole (A balancing game) to it.

While I was looking more into the logging side of the code I found out how to get the PPO agent to properly log their training data. Because of that I ran the code again but this time I added the PPO agent to it. While this increased the training time it didn’t do so drastically because of the Cuda software.

# Data Visualization

As I have said before, I let 3 agents learn in 5 different environments in order to collect data about their learning rates. Each agent was fresh and thus did not learn anything beforehand. What will be shown in this document is the average episode length and the average reward that the agents get. There are more values that I can get out of the data, however those values aren’t all that useful. In most cases only one agent will log different values (Learning rate, policy loss, value loss, etc..) or I get broken data from those values. Since the only to value that all agents log is the average reward and length, those are the graphs I shall be showing and comparing.

Since Tensorboard is the baseline for plotting with stable baselines, that is what I used. A limitation of Tensorboard is that I can not change much about the plots that are made. I can’t set or label the values of the Y-axis and the X-axis only has three options those being: Timesteps, Relative time and wall. Timesteps will show the amount of steps that the agents went through. Relative time shows the training length in minutes and wall shows the duration of the training starting with the real time start time and ending in the real time end time. To still try and make clear what the graphs show, the explanation of the graphs shall be shown in the by script of the graphs. I will also use the same colours for each agent. The colour listing is as follows:

* A2C Agent is shown in Red
* DQN Agent is shown in Blue
* PPO Agent is shown in Green

I shall be using the colour codes through the chapter in order to keep clear which line is which agent. There is not much that can be done in showing the difference between the agents when one is colour blind without showing the results directly in Tensorboard. For that reason, this chapter if printed must be done in colour it would be impossible to show the difference between the lines in the graphs.

I learned too late that it is possible to overwrite the default logger and log everything in a CSV file. I did not have time to train all the agents again and learn how to use Pandas and Mathplot and make my own plots, using the three different CSV files in the same plot. Trying to merge the files results in plots that are far longer and with a lot of outliners. They also don’t show the difference between the agents, as the plot believes everything to be part of the same agent.

## Breakout Visualization

I started with Breakout in the data preparation chapter and thus I start with Breakout in the visualization. I shall be following the order of average reward being shown before the average length of the episode. Figure 6 shows the first graph showing the average reward that the agents got in the breakout game. As said before, the by script shows the X- and Y-axis values.

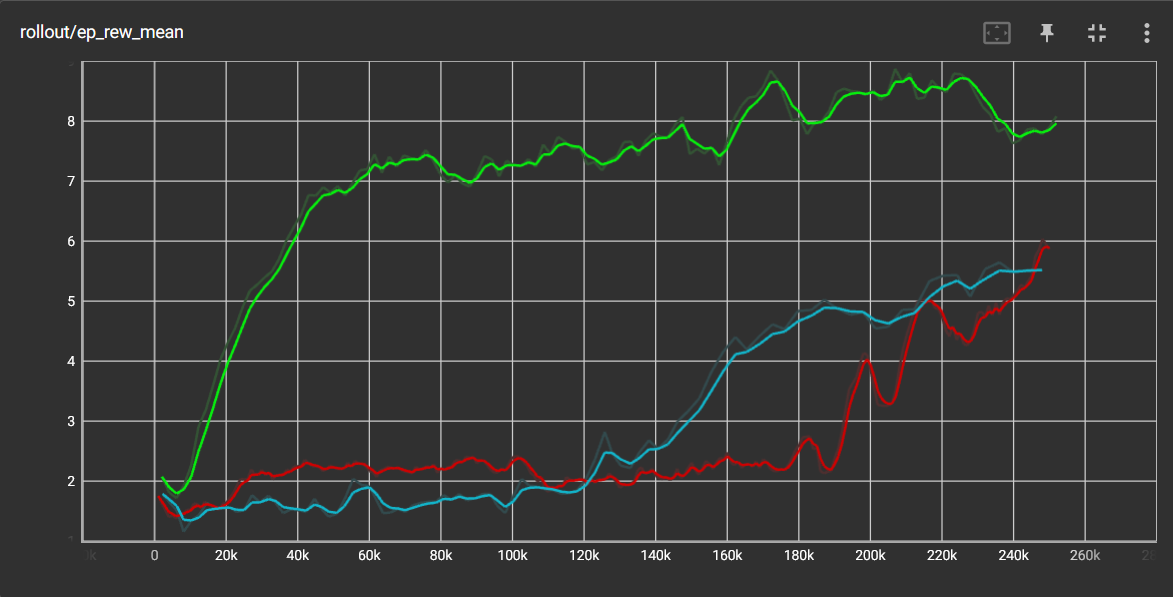


Figure 6 Average Reward breakout (Y-axis shows average Reward per 100 episodes & X-axis shows Total Timesteps)

Starting with the A2C-Agent it can be seen that the agent needs some time to truly get going. It’s only around the 180k Timesteps that the agent learns something in order to get more points. After that they start growing far quicker then before. The DQN-Agent also needs some time to get going. They are, in the beginning, worse of then the A2C-Agent but then around the 120k mark they have found a way to get more points. They quickly grow before evening out again around the 200k mark. After that, is a slower but steady growth. The A2C-Agent at that point is starting to surpass the DQN-Agent. Lastly we have the PPO-Agent. This agent has shown that it can quickly learn and find ways to get the highest reward as fast as possible. From the ~8k mark until the 60k they quickly grow and leave the other agents in the dust. After that, they even out and there is not much growth left. Since all agents are trained until the 250k mark, I can only speculate on what happens next. From the graph it would look like the A2C-Agent will keep growing, while the DQN-Agent might keep it’s slow but steady growth or might even out and stop growing. There is a good chance that the PPO-Agent will keep an even path. Not truly growing but also not falling below what it has.

The PPO-Agent is, at this point in time, the best agent for the breakout game. As they have been able to get the highest score in the allowed time. That of course does not mean that it will always be the best of the three agents for the Breakout game. It’s definitely possible that the PPO-Agent will not grow any further. So it could be that in another 250k or more step steps one of the other agents will rival or surpass it. But for short training session the PPO-Agent is far better then the other agents.

The graph of figure 7 will show the mean episode length of the agents. The mean episode length on the Y-axis is an unknown unit of time. This would normally make it hard to read the graph as you don’t know what you’re looking at. Luckily my goal is to compare and so, even when not knowing what the value of the Y-axis is, I can still see the difference between the agents.

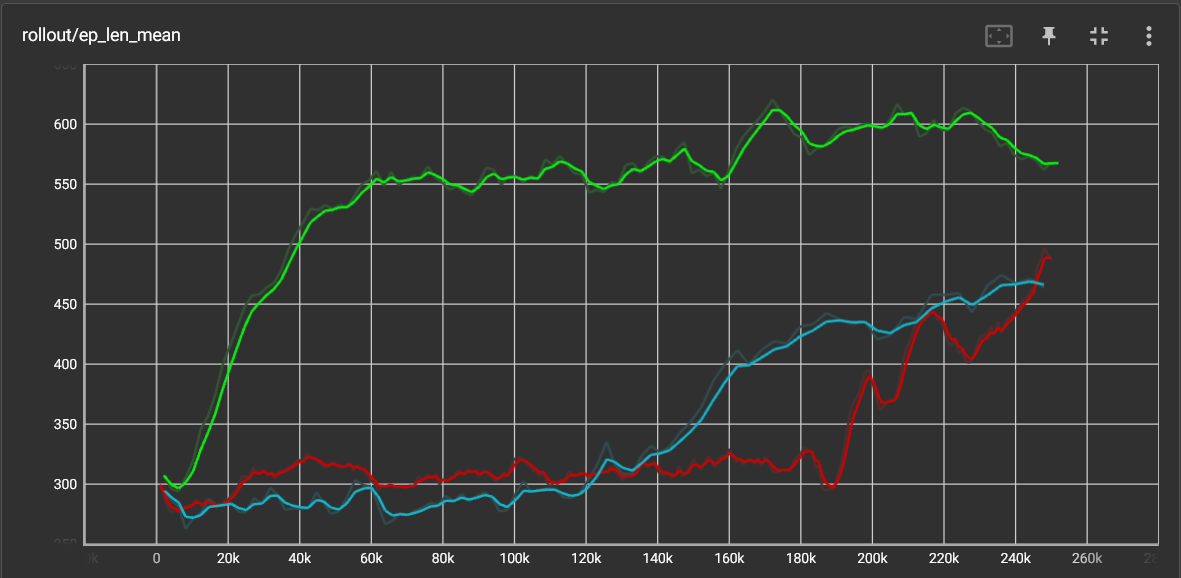


Figure 7 Average episode length breakout (Y-axis shows the mean episode length in an unknown unit & X-axis show the total timesteps)

Figure 4.2 looks a lot like figure 4.1. In order to get rewards in the Breakout game, the agent must destroy the coloured blocks. The longer the agent stays in the game the more blocks they can destroy and so it not wrong to say the two graphs will follow each other. There is a definite connection between the rewards gained and the amount of time that the agent stays in the game. This is to be expected as staying longer in the game will result in more accidental points.

## Pong Visualization

The colour codes and the values of the X- and Y-axis stay the same as is said in the preface of chapter 4. Figure 8 shows the rewards the agents got during the pong game over an average of a 100 episodes.

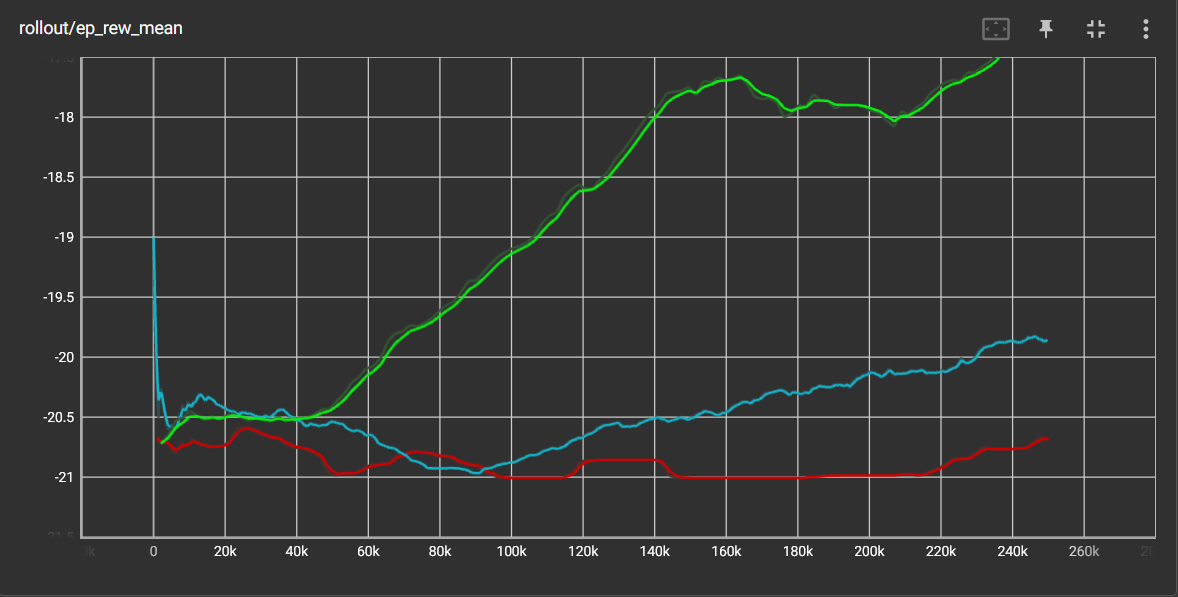


Figure 8 Average reward Pong game (Y-axis shows reward & X-axis shows total timesteps)

The first thing that becomes obvious is the agents are all scored in the negative. This is because the pong game gives negative scores when the opponent makes a point. As such it can be said the agents aren’t good at this game, after training for 250k timesteps. Outside of the outliner from the DQN-Agent at the very start, all agents start at the same point. The A2C-Agent starts at a better point then it ends. There is no growth in the A2C-Agent, in fact it only start falling further away from a better score. The DQN-Agent however does better, though not by much in the start. The DQN-Agent needs a moment to get going and, at the 90k mark, shows that it learned something. After that point the DQN-Agent has a stable growth that doesn’t show signs of evening out. Once again the best agent is the PPO-Agent. Just like with the other agents, it takes around ~50k timesteps before it learned something useful but when it does, it starts growing fast. The growth of the agent peaks for the moment at the 160k mark, before falling down a bit. This fall can be attributed to the agent trying something new for a few episode before learning that it doesn’t work better and going back to what works and starts growing again.

Once again the PPO-Agent Is the best agent within the limits set by the training time. However unlike with the breakout game, the DQN-Agent beats the A2C-Agent.

Figure 9 shows the average episode time of the agents during the pong training.

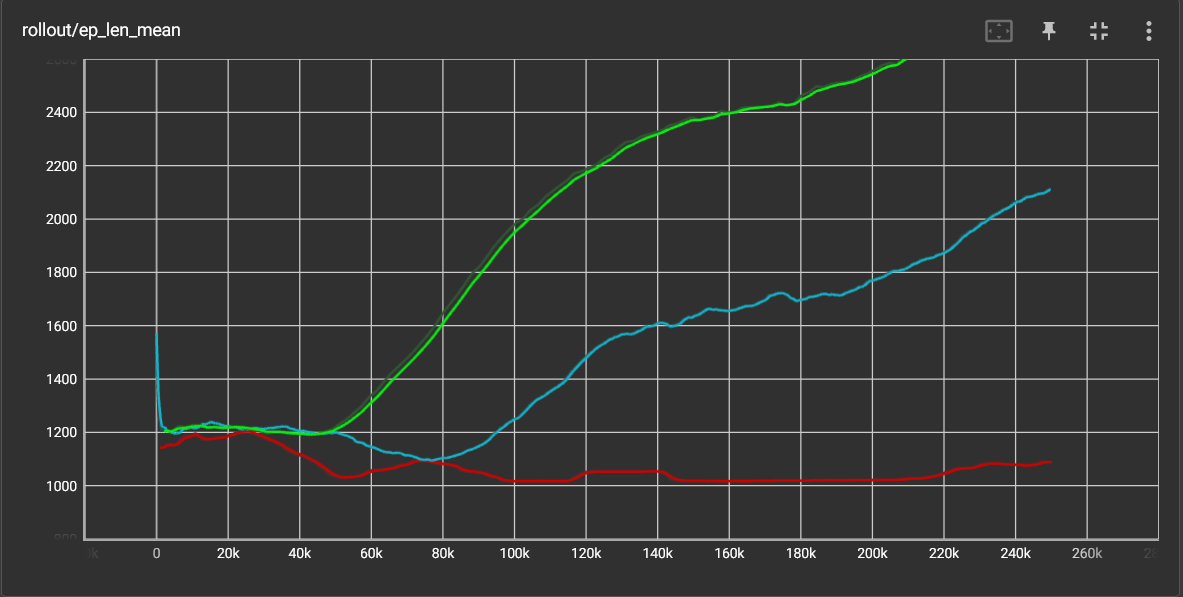


Figure 9 Average episode length pong (Y-axis shows the mean episode length in an unknown unit & X-axis show the total timesteps)

When putting the graphs next to each other, it can be seen that they once again looks similar. The outliner in figure 8 for the DQN-Agent is also visible at the start and all the agents follow roughly the same path as in figure 8. That small dal that can be seen in figure for the PPO-Agent Is also shown in the graph of figure 9 as around the 160k mark the time average evens out a bit before growing longer again.

## Blackjack visualization

Next game is going to be Blackjack. It is something very different from the previous two games and thus results are also very different. Figure 10 shows the average reward for the blackjack agents.

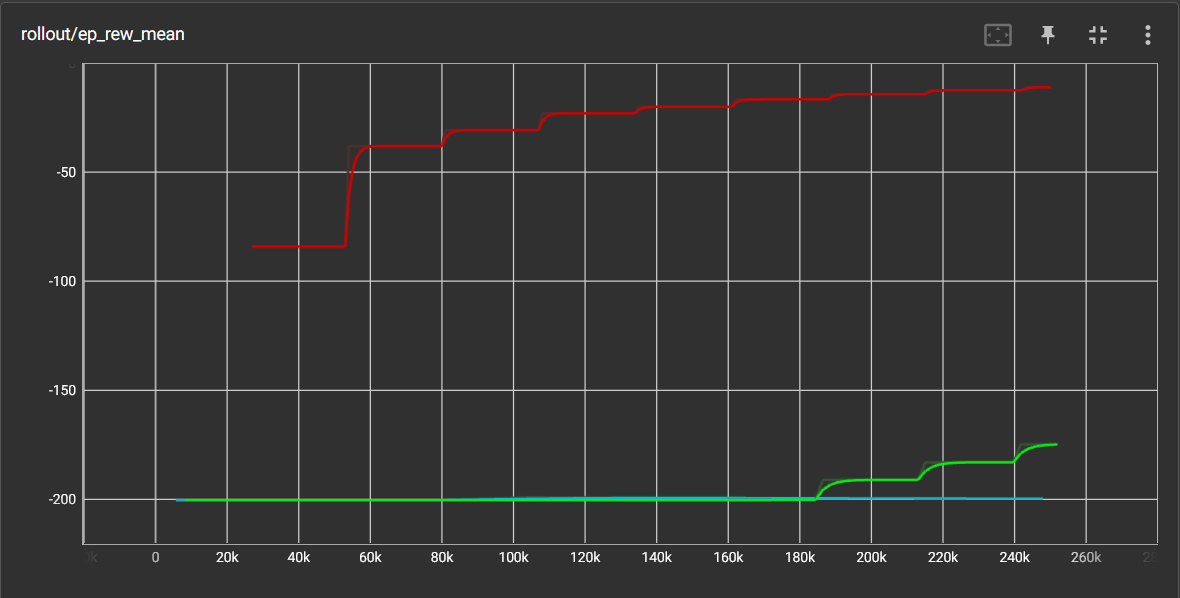


Figure 10 Average reward Blackjack game (Y-axis shows reward & X-axis shows total timesteps)

A big difference can be seen between figure 10 and previous reward graphs. The starting point of the A2C-Agent lies far higher then with the other two agents. It also becomes obvious fast that the A2C-Agent is the best agent for this game withing the limits of the training. The A2C-Agent shows a steady growth, though it does become less and less the further we’re going into the game. The DQN-Agent seemingly isn’t able to play well. While it isn’t fully visible in figure 10, the DQN-Agent started at -200 but ended at -199. The PPO-Agent, which has so far been the best agent, need quite a bit of time before it learned something useful. After it started learning at the ~185k mark, it shows a growth curve similar to the A2C-Agent. Blackjack is a pure counting game and it seems that this is what the A2C-Agent excels at. However as I said before, the A2C-Agent shows signs of evening out near the end. It possible that it would stop learning and growing better in another 250k time steps. At that point it could be possible that either the PPO-Agent or the DQN-Agent surpasses it.

Just like the average rewards for the blackjack agents was far different from the breakout or pong agents, so is the average episode time also very different. Until now, the average time has followed a similar form as the average rewards. However figure 11 shows something entirely different.

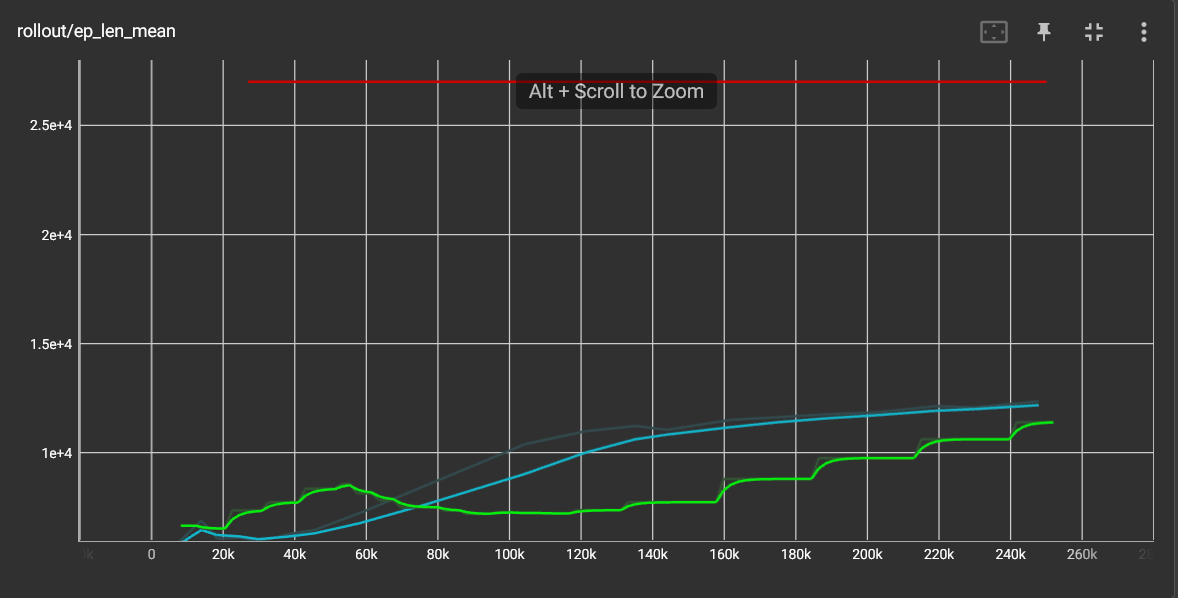


Figure 11 Average episode length blackjack (Y-axis shows the mean episode length in an unknown unit & X-axis show the total timesteps)

Because of the great difference between the A2C-Agent, DQN-Agent and the PPO-Agent, Y-axis scale is also far greater then with the previous agents. It can be seen the A2C-Agent has been very static in their episode time, while the other two agents show difference. DQN-Agent which the least growth in terms of reward does so some difference in the time it spent in each episode. Starting low until it spent more time in the episodes then the PPO-Agent. In the other two games, a longer episode time also meant a higher reward. That was because, just being in the game allows one to get points by accident. With blackjack that isn’t possible. If the agents don’t do anything then they will just lose when their time runs out. The DQN-Agent does show that it starts doing something, but it isn’t enough to increase their reward. The PPO-Agent does end up showing a similar form between the average time and the average reward as the A2C-Agent.

## Taxi Visualization

The graph of figure 12 shows that results of the blackjack training wasn’t an outliner. Once again the A2C-Agent shows that it is better then the other agents, within the limit set.

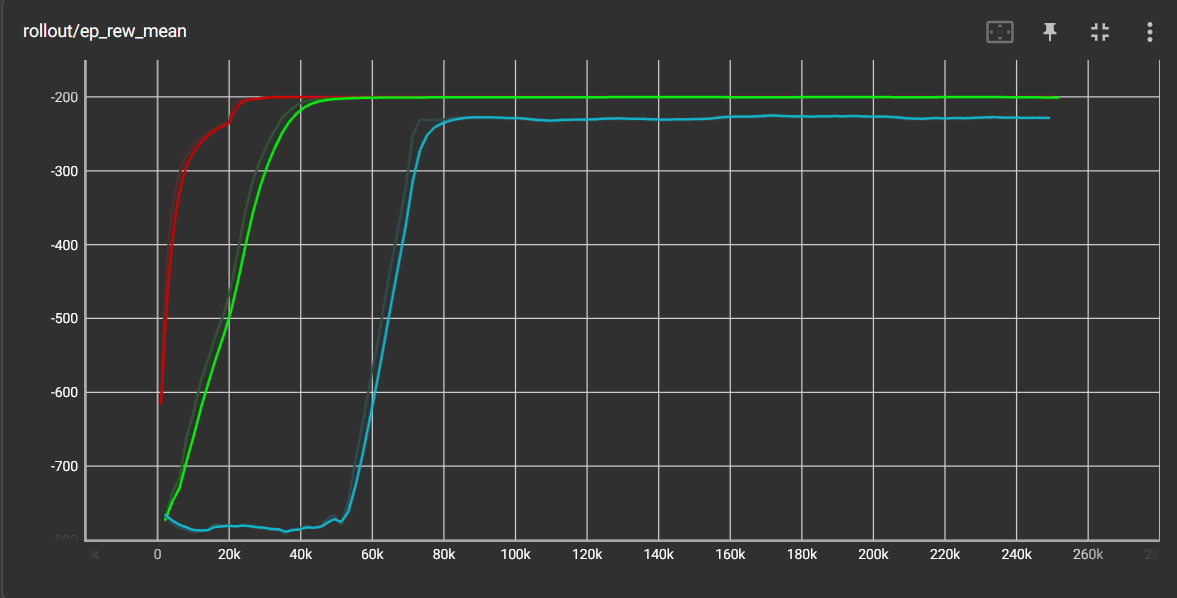


Figure 12 Average reward Taxi game (Y-axis shows reward & X-axis shows total timesteps)

Between the start time of the training and the first log, the A2C-Agent has been learning and growing. It can roughly be said that all agents start at around -780 points. The way the rewards work for the taxi game is that every time the ‘pickup’ or ‘dropoff’ action is perform in a location where it should not be, the agent get -10 points. Also for every step the agent makes, that does not invoke a different reward, they get -1 point. Every time the passenger is successfully dropped off the agent gets +20 points. This is something that the A2C-Agent and the PPO-Agent seem to learn. However when both agents get to -200 points, they decide that enough is enough. This is the best they are going to get and they decide not to try anything else. The DQN-Agent, true to form, needs a moment to get going and gets the same growth curve as the other agents before stopping at -220 points. From how the graph looks, it doesn’t seem like there is any reason in having the agents train more, as they don’t show any growth in over 200k timesteps.

Figure 13 shows the average time for the Taxi game.

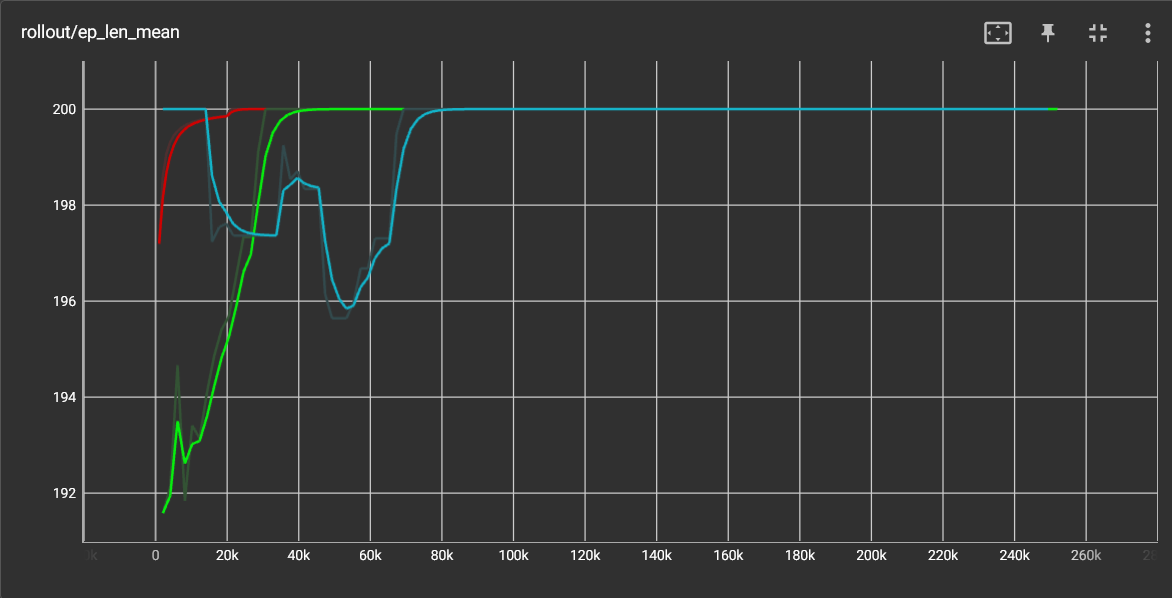


Figure 13 Average episode length Taxi (Y-axis shows the mean episode length in an unknown unit & X-axis show the total timesteps)

The first thing that becomes clear is that all three agents end up at 200. Surprisingly the DQN-Agent was the first to get there at the very start before dipping down. At the time it starts growing again, you can also see the increase in reward shown in the rewards graph. The other agents show the rapid increase that have also shown in the rewards graph. The Taxi game only allows a maximum episode length of 200.

## Pole keeper Visualization

Pole keeper is the last game and figure 14 shows the average rewards for the game.

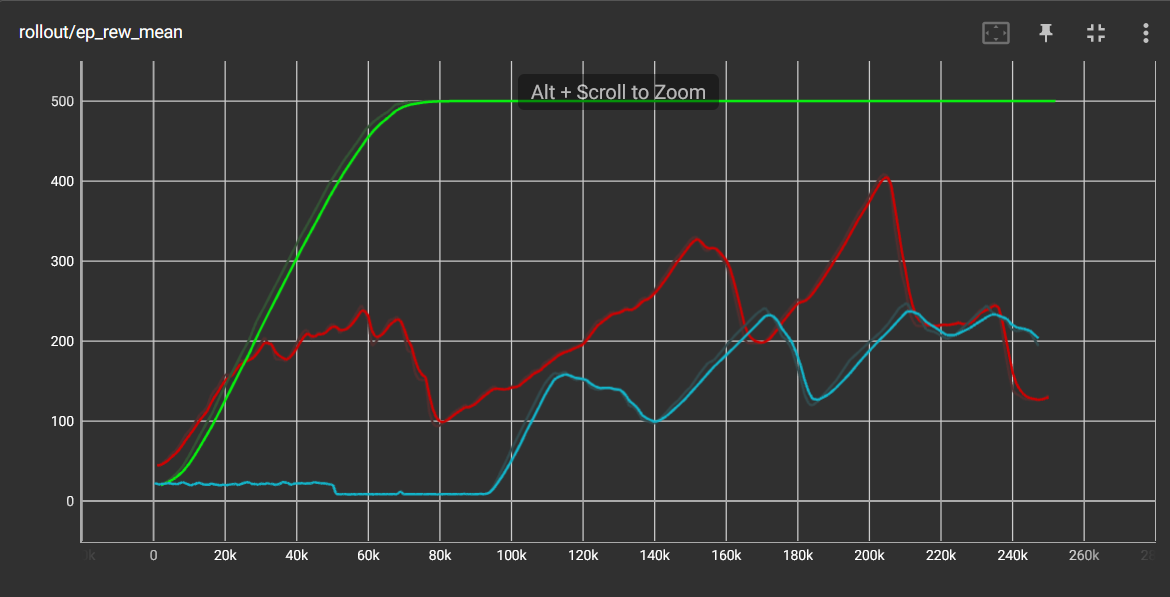


Figure 14 Average reward Pole game (Y-axis shows reward & X-axis shows total timesteps)

Starting with the A2C-Agent it shows a good start, before dipping down and slowly climbing up again. This repeats a few time as the A2C-Agent is trying out new things and figure out what works. However at the end of the training they are nearly 300 points beneath their peak. The agent did show that they are learning and didn’t seem to have given up as it was in a few of the previous games. The DQN-Agent is still true to form and takes a while to get going, but when it does it grows faster then the A2C-Agent. At the end of the training the DQN-Agent has even surpassed the A2C-Agent. The true winner of this is the PPO-Agent. The way the points work in this game, is that the agent has to survive as long as possible. The PPO-Agent has seemingly reached the maximum reward possible of 500 points. The PPO-Agent did this at around 1/3 of the total training time.

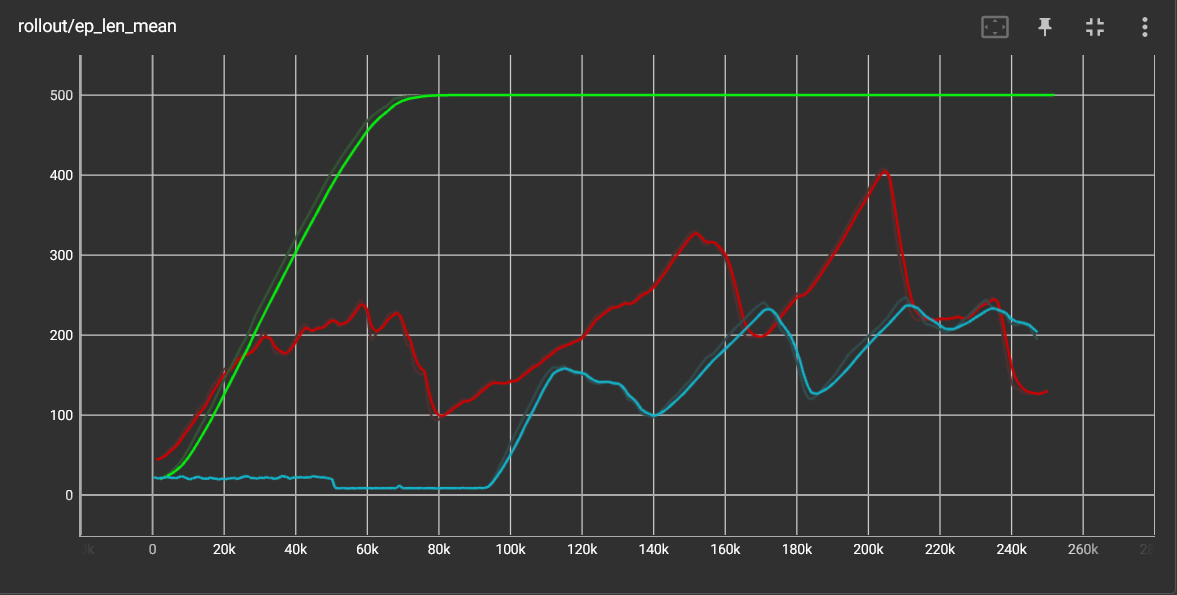


Figure 15 Average episode length Pole keeper (Y-axis shows the mean episode length in an unknown unit & X-axis show the total timesteps)

Since the reward is depended on the time alive in the game, it is no surprise that both graphs look just like each other.

## Conclusion

Each training log was made with a fresh unlearned agent and in most cases the results speak for themselves. The PPO-agent, was the best performing in 3 out of 5 games and the A2C-agent performed best in the remaining 2 games. I can’t say why they are the best without going into the algorithm itself and the theory behind it.

There were cases in which the PPO-agent won, but was stagnating at the point where the training stopped. So if given a longer training time then results may differ. As at some times some agents showed signs of stagnating while other were growing steadily. Thus everything that is said, can only be said within the limits of the training time.

For most games it does seem like the PPO-agent is the best choice to go with if you want to quickly train an agent to be able to play. Followed by the A2C-agent and then the DQN-agent. Exceptions to this are counting games like Blackjack where the A2C-agent is far better. The games themselves can be divided into Atari games, Text games and Control games. For both Atari games and Control games, it is the PPO-agent that you would want. For Text games it’s the A2C-agent. This is again only so for shorter training session as longer training session might bring different results.

# Closing words

When I started writing this document I only had 2 agents and 4 environments. During the course of me writing this document, there were some changes in my personal project. I manged to get the third agent to work and I ended up using an extra environment in something totally different from the other environment to see more difference. The changes made it so that my conclusion and my thoughts about the challenge changed. The appearance of the new agent (PPO) made it so that the first winner (A2C) suddenly as less good. I also changed the original training time form 500K time steps to 250K timesteps in order to lower the training time for the new sets of training logs. This also had some influence on the endings. But all-in-all, I am satisfied that the new agent did so well and the graphs do show that interesting difference that I wanted to see.

I would also like to thank: Georgiana, Hans and Coen for their help during the course of this project. When I was stuck on certain parts, they managed to help get through it. It would have been far harder to bring this project to an end without them.