CS 4200 Final Project

Project Report

Deliverable 4

Aryan Miriyala

Cedric Dortch

Dr. Shuteng Niu

April 21, 2024

Introduction

This project helps us better understand Reinforcement Learning Algorithms and how to utilize them to have agents understand their environment and take decisions based on the situations. We will try to develop an Artificial Intelligence agent that can play and clear the first level of Super Mario Bros by using reinforcement learning techniques. The agent will have access to limited controls and the environment will be defined for the agent. The action space is going to be limited to only a few movements, and we will also try to define the state space for the agent and the environment. The goal of the project will be to get the agent to converge to state where it will be able to clear a level and we are targeting the first level of the game. Employing the specific algorithms is going to be a challenging task, considering the dynamic environment of the game and the complicated nature of the numerous challenges that will be occurring throughout the game.

The project describes two specific reinforcement learning algorithms that will help us achieve this task. We will be using Deep Q Network (DQN) and Proximal Policy Optimization algorithms, which are specific reinforcement learning algorithms. PPO utilized a small step size, blending the aspects of A2C (Asynchronous Advantage Critic) and TRPO (Trust Region Policy Optimization) to ensure the agent is steadily moving towards the best solution. On the other hand, DQN draws inspiration from Filtered Q-Iteration and incorporates various learning strategies to enhance learning stability with neural networks. It employs a replay buffer, a target network, and gradient clipping technique.

The objective is to train our Mario agent with both the algorithms and evaluate their respective strengths and weaknesses to determine which algorithm is superior for training the Mario agent. It depends on training capabilities, effective speed, and generalization.

Related Work:

Developers have made previous efforts to utilize the above-mentioned algorithms to achieve the task of training a game agent to learn the rules of the game and play the game by itself.

The report by Stanford students Yizheng Liao, Kun Yi and Zhe Yang (Google) talks about applying reinforcement learning algorithms to design an automatic agent to play the game Super Mario Bros. They describe the same challenges that we have mentioned above about the handling of the environment. They try to abstract the game environment to a state vector to simply the complexities and use Q Learning to achieve fast convergence. I have referenced the journal article below.

[1] Y. Liao, K. Yi, and Z. Yang, "CS229 Final Report Reinforcement Learning to Play Mario." Available: https://cs229.stanford.edu/proj2012/LiaoYiYang-RLtoPlayMario.pdf

Another paper by Sergey Karakovskiy and Julian Togelius describes the benchmarks for reinforcement learning algorithms and game AI techniques developed by programmers and summarizes these contributions and gives an overview of the state of the art in Mario playing AI and talks about the development of these benchmarks.

[2] S. Karakovskiy and J. Togelius, "The Mario AI Benchmark and Competitions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 55–67, Mar. 2012, doi: https://doi.org/10.1109/tciaig.2012.2188528.

Nicholas Renotte:

Nicholas Renotte demonstrates the use of PPO algorithm to train the Mario agent by using the OpenAI's gym framework, and numerous other tools to achieve what he has described. We drew inspiration from his work and have implemented it in our own. The rights for the

source code belong to the rightful owner and we have modified it to fit with our needs. The video link for his demonstration and a GitHub link to his source code is provided below.

[3] "Build an Mario AI Model with Python | Gaming Reinforcement Learning," *www.youtube.com*. https://www.youtube.com/watch?v=2eeYqJ0uBKE&list=PLFNrAHO8tpSyv64czmtRuLwd9hrNCuxPp&index=2 (accessed Apr. 17, 2024).

[4] N. Renotte, "nicknochnack/MarioRL," *GitHub*, Oct. 02, 2023. https://github.com/nicknochnack/MarioRL

Sourish Kundu:

Sourish implements DQN, precisely DDQN (Double Deep Q Network) algorithm to achieve convergence for his Mario agent. He tries to build this model completely from scratch using NES. He does a good job at introducing the vocabulary we are going to use for this refers to important papers for information. We use his tutorial on DDQN algorithm as an inspiration to implement our DQN algorithm to train Mario more efficiently. All the code and intellectual knowledge is owned by Sourish, and we have used bits and pieces to implement our own model.

[5] "Train AI to Beat Super Mario Bros! || Reinforcement Learning Completely from Scratch," *www.youtube.com*. https://www.youtube.com/watch?v=_gmQZToTMac&list=PLFNrAHO8tpSyv64czmtRuLwd9hrNCuxPp&index=4 (accessed Apr. 17, 2024).

[6] "Super-Mario-Bros-RL/README.md at main · Sourish07/Super-Mario-Bros-RL," *GitHub*. https://github.com/Sourish07/Super-Mario-Bros-RL/blob/main/README.md (accessed Apr. 17, 2024).

Medium.com:

This particular article from medium.com talks about training a reinforcement learning agent to play the game Super Mario Bros, which is the topic that we are trying to tackle in our project. The writer discusses about using DQN and DDQN algorithms to achieve this task. There is discussion about gray scaling, resizing, frame skipping, and frame stacking – most of which we are going to employ into our project as well. The topics that we discuss and the strategies that we use in our project are inspired from this article and the videos I have posted above.

[7] S. Padhye, "Playing Super Mario Bros. with Reinforcement Learning," *Medium*, Nov. 27, 2022. https://sohum-padhye.medium.com/playing-super-mario-bros-with-reinforcement-learning-81ee0c235372 (accessed Apr. 17, 2024).

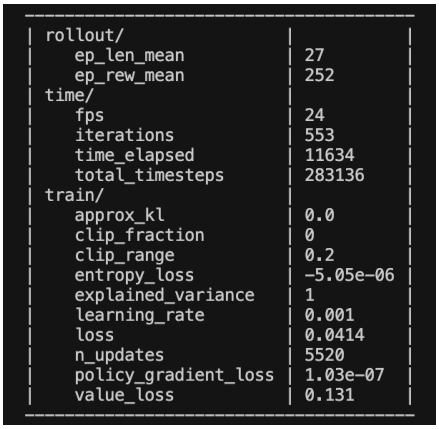
Approach

Problem Analysis:

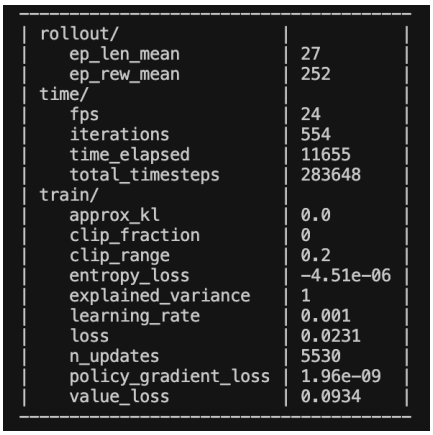
Prior solutions to the Mario agent using reinforcement learning have been published before and they primarily use either Proximal Policy Optimization (PPO) or Deep Q Network (DQN) or Double Deep Q Network (DDQN) to achieve desired results. Let's breakdown these two processes and see their strengths and weaknesses and why it might be better to use one over the other.

Proximal Policy Optimization (PPO):

PPO belongs to the family of policy gradient methods, where the agent is learning through a parameterized policy and directly manipulating it to optimize the expected result. The main goal is to maximize the probability of good actions and to minimize the use of bad actions. Because it tends to exhibit better sample efficiency, it might be converging faster, making it a very valuable resource in cases where we need to collect data and this is more valuable.



rollout/	
ep_len_mean	27
ep_rew_mean	252
time/	
fps	24
iterations	553
time_elapsed	11634
total_timesteps	283136
train/	
approx_kl	0.0
clip_fraction	0
clip_range	0.2
entropy_loss	-5.05e-06
explained_variance	1
learning_rate	0.001
loss	0.0414
n_updates	5520
policy_gradient_loss	1.03e-07
value_loss	0.131



rollout/	
ep_len_mean	27
ep_rew_mean	252
time/	
fps	24
iterations	554
time_elapsed	11655
total_timesteps	283648
train/	
approx_kl	0.0
clip_fraction	0
clip_range	0.2
entropy_loss	-4.51e-06
explained_variance	1
learning_rate	0.001
loss	0.0231
n_updates	5530
policy_gradient_loss	1.96e-09
value_loss	0.0934

The above two images are samples from running the PPO algorithm on my machine to train the Mario agent. From both these images, we can see that the algorithm has gotten better over time. The gradient loss in the first image is much larger than the second video and the value loss in the first image is larger than the second image as well. From these instances, it is clear that the PPO algorithm is getting better with time. The learning rate is set by us and is changeable. We have set the value to be relatively high, which means it is not in the fine tuning phase, but we do not have machines that can handle low learning rates and generate an effective policy, so we had to decide to go with this learning rate. The entropy loss is in negative as well, which could indicate that the model is converging to a deterministic value. I have referenced the article that helped us with understanding the values and what they mean below for the reader's reference.

[8] D. P. Teh, "Playing Super Mario with PPO Reinforcement Learning Agent," *Medium*, Feb. 16, 2024. https://medium.com/@darioprawarateh/playing-super-mario-with-ppo-reinforcement-learning-agent-e9f61fd04b32

Deep Q Networks (DQN):

DQN is based on Q-Learning, a value-based learning method. It learns to approximate the Q values of state-action pairs which enables the agent to select action that maximize the cumulative reward. A setback for DQN is that it might require a greater number of iterations to converge compared to PPO because of its dependence on experience replay and off policy learning.

In the context of a Mario agent, theoretically it seems like DQN would be a better choice because of its standards for stability, sample efficiency and suitable environment. The action space that we are going to employ for our project aligns well with DQN's architecture, where Q values are updated after every output. But we also need to consider the computation space and resources available when considering such an act.

Resources Used:

We used numerous online resources and have taken inspiration from many GitHub repositories to achieve the task of training a Mario agent with two different algorithms and getting the results for both and comparing them to decide which one to go with. Theoretically, DQN should serve as a better option, but it heavily depends on the action space, resources, and multiple other things.

I have referenced the GitHub repositories here:

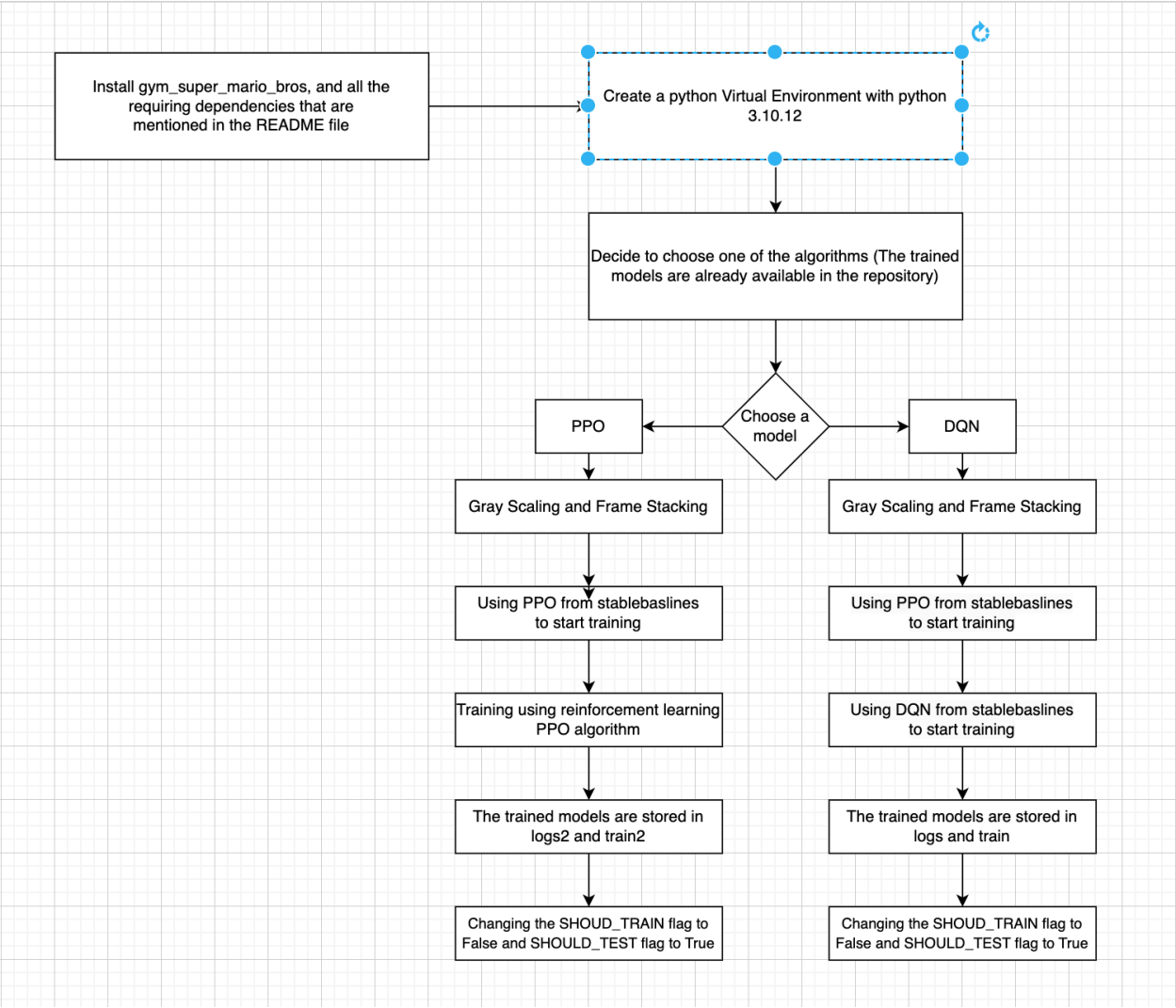
[1] N. Renotte, "nicknochnack/MarioRL," *GitHub*, Oct. 02, 2023. https://github.com/nicknochnack/MarioRL

[2] "Super-Mario-Bros-RL/README.md at main · Sourish07/Super-Mario-Bros-RL," *GitHub*. https://github.com/Sourish07/Super-Mario-Bros-RL/blob/main/README.md (accessed Apr. 17, 2024).

[3] S. Padhye, "Playing Super Mario Bros. with Reinforcement Learning," *Medium*, Nov. 27, 2022. https://sohum-padhye.medium.com/playing-super-mario-bros-with-reinforcement-learning-81ee0c235372 (accessed Apr. 17, 2024).

[4] Y. Feng, "yfeng997/MadMario," *GitHub*, Mar. 26, 2024. https://github.com/yfeng997/MadMario (accessed Apr. 17, 2024).

Software Design:



This flowchart described how our software functions and what it does to achieve the trained models and to execute the logs. We used numerous sources and took inspiration from those

sources to fix on our trained model. We decided to go with DQN and PPO because they seemed to be the most widely used models for AI training in terms of game training AI models. We implemented our DQN.py and PPO.py files inspired from Nicholas Renotte and Sourish Kundu, who have implemented the said algorithms in their videos and repositories to train their own AI models for a Mario agent. They have described in their video that training such AI model with a huge action space requires a lot of time and computation and it took them days to do it. We have an agent_nn.py file which is the neural network file for the agent, that is required by both the algorithms. The logic for this is borrowed from Sourish's GitHub repository and so is the agent.py file. These are the two files that deal with the training of the agent and building the neural network that we are going to use to train Mario and help him clear the level, or at least get close to doing it. We connected resources from Nicholas and Sourish and used various ideas from both the developers to get the model that would fit with our needs and help us with the build of the trained model. We use concepts of gray scaling and frame stacking – gray scaling hopes to convert the rgb model of the game to a gray scale effectively cutting the data to one-third of its actual data which makes it easier to process and collect. Frame stacking is a technique of stacking different frames together, creating a concept of memory for our Mario agent, and helps him make decisions and to look futher for obstacles and enemies. We have borrowed and learned these concepts from online sources.

Source Code Description:

File Name	Function
preprocessing.ipynb	This is a Jupyter Notebook file that helps us with the preprocessing of the environment. A lot of concepts within the file are borrowed from Nicholas's Mario AI agent program. GitHub: https://github.com/nicknochnack/MarioRL/blob/main/Mario%20Tutorial.ipynb We use various concepts of gray scaling and frame stacking here.
PPO vs DQN README.md	A readme file describing the use of two algorithms that we have used to train our Mario agent. We describe the algorithms and provide an insight and showcase some of the functions and libraries we need to execute properly.
sample-mario.py	The very first iteration of our sample environment to run the Mario game UI using python.
logs and log2	These folders contain information about our iterations and total steps for training the agent. It takes in information that could be later used to represent quantifiable logs, which could be converted into graphs that we will be presenting in our report.
train and train2	These folders contain the trained models we have for each of the algorithms. The models are saved with a frequency fo 10,000 steps meaning a new model is created for every 10,000 steps taken by the agent so far.
agent_nn.py	This is the python file that creates the neural network that we will be using for the trained models and to run the game. This helps the AI agent to process data and defines the classes that build a neural network, we're using convolutional neural networks (cnn) with 3 convolutional layers with 2 linear layers.

agent.py	This file creates an agent class. We will import our neural network and replay buffers. The number of actions is given, and a tracker keeps a check on the number of iterations the agent has been through and what it has learned. We define the hyperparameters here as well and pass, the learning rate, epsilon, discount factor and others.
DQN.py	The file implementing and using the DQN algorithm. It utilized different tricks to stabilize the learning with neural networks. We import multiple libraries and wrap our agent and environment in multiple wrappers discussed elsewhere. We create our agent object and pass in various parameters. We define the episode and actions and store the information for each step. We are also training our agent here.
generate_clips.py	We are utilizing the concept of skip framing and frame stacking here. Skip frame uses an action taken in a given frame and repeats it after a given set of frames, while training to maximize the reward function. Frame stacking utilized the concept of multiple frames and given the agent a sense of memory so it can take necessary actions and so the agent is aware of the environment so it can take the viable action for a given state.
How-to-run-code-README.md	A readme file describing the actions needed to take and the libraries that need to be installed before being able to run this program.
PPO.py	The python file that implements and utilized the PPO algorithm into our Mario agent. The algorithm deals with policies and gradients and tries to train the agent to utilize the PPO algorithm. We import multiple libraries and wrap the game into the joypad wrapper. We also utilize multiple wrappers here. It is very similar to the implementation of DQN but with a completely different algorithm altogether.
requirements.txt	To install all the required libraries and packages needed to run the program successfully.

utils.py	Miscellaneous utility functions that calculate some needed time and date variables as necessary. Implementations for mean and average are provided as well.
wrappers.py	This python file contains all the wrappers that we will be using for our implementation. We are using skip frames, gray scaling, frame stacking and resize wrappers for our code and you can find the implementation for each of those here.

Evaluation and Results

Let's talk about the evaluation of these algorithms and the results they produced with the training they have received. It must be noted that the people that were able to successfully train their models to pass the first level of the game and superior computing power, often had access to graphics cards, and computers that were able to handle such intense computation for multiple days in a row. The step size for the PPO algorithm to converge had to be around 4 million, and that is how Nicholas was able to get the agent to pass the level and for the model to converge. The iterations for DQN algorithm were around 50,000 before the model was able to converge, and Sourish, who handled the DDQN algorithm was able to achieve this with the help of RTX Graphics card which ran the training for days, before he was able to get the model to converge and get the Mario agent to pass the level.

After that has been said, the computers that we had access to did not support such huge computations, and we were forced to have models that pseudo converged, so we were not able to get the agent to clear a level successfully, but we can demonstrate that the agent is learning new strategies with each iteration and is trying to take better steps.

Let's look at the PPO Algorithm:

PPO looks at the problem in a policy defined standpoint. It tries to optimize the policies used for deciding the next steps based on the best actions to take while in the current state. I have trained the model up to 100,000 timesteps and it seems to have gotten a good idea about how to clear obstacles and how to engage with the environment. Let's start with defining the states and the action space. The states are numerous with Mario looking to pass the game by using a skip frame method which allows him to use actions for frames with a variable length. The action space is restricted to SIMPLE_MOVEMENTS, which includes going right, jumping, double jumping and doing nothing. We did this to reduce the load of actions and to reduce the number of state-action pairs that could be generated. The game itself is very intense and it is very difficult to record every state action pair and to build an optimal policy for those pairs.

An important step to help our reinforcement learning model to learn about its environment and to be able to make decisions is to supply it with the correct data. To do this, we will use two preprocessing methods – gray scaling and frame stacking – to help collect this data. Converting the game to gray scale, helps with reducing the amount of data to learn from because now it doesn't have to deal with rgb colors. Frame stacking helps our AI with context, by stacking frames on top of each other by effectively helping us give our model the aspect of memory, and for the agent to see the movements of the enemies and the dynamic changes in the game environment.

Let's talk about Vectorization models, which are needed to build our reinforcement learning algorithms. We are going to be using stablebaselines, which was originally built by OpenAI. We are going to work with the stacked environments and some dummy vectors so we can pass the information we collect from the frames onto our AI model. StableBaselines is a reinforcement learning library that helps us with already available reinforcement learning

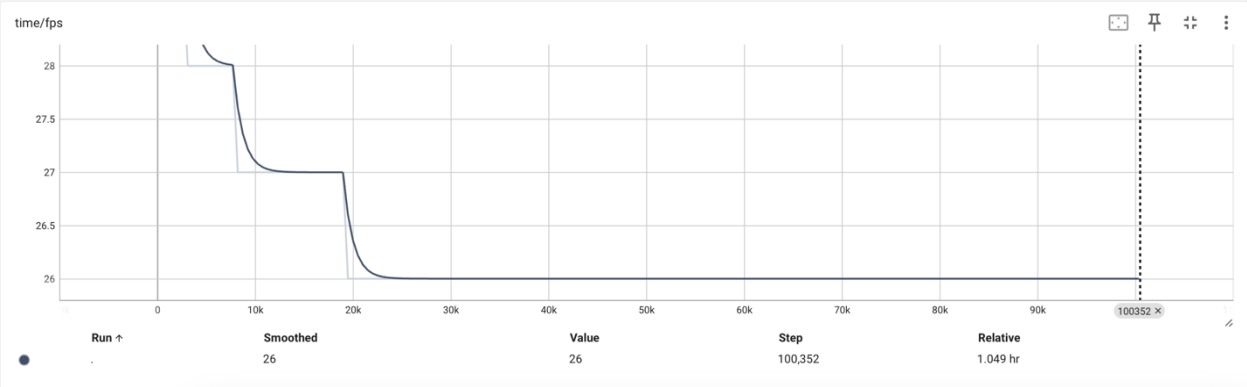
algorithms that we would use and will be using to achivev our desired task. We will also need to install PyTorch to be able to use these packages. Installing PyTorch is specific to devices, so we will need to look at their website before we go on to install them.

[1] "PyTorch," *www.pytorch.org*. https://pytorch.org/get-started/locally/

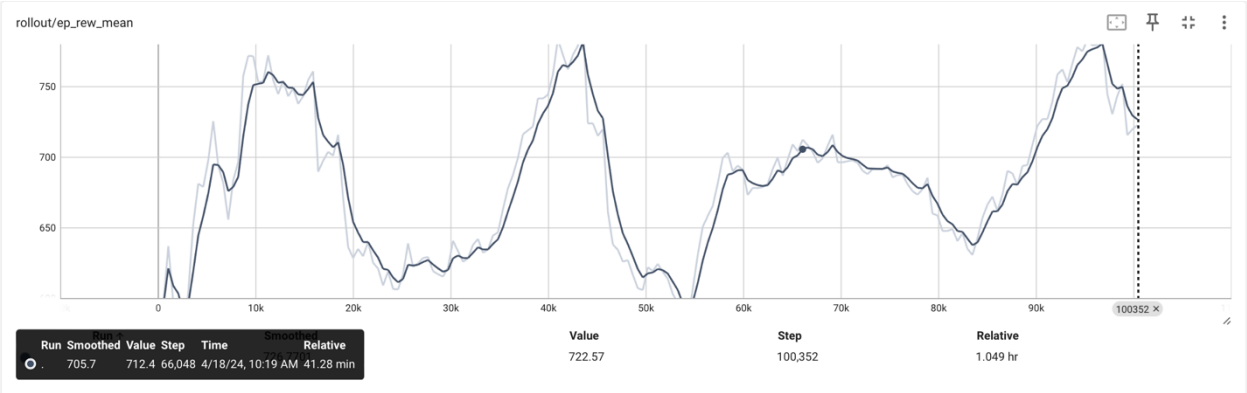
I have included a document that talks more deeply about what PPO is and how it functions here.

[2] "PPO — Stable Baselines3 1.4.1a3 documentation," *stable-baselines3.readthedocs.io*. https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

Let's talk about metric and evaluation of our algorithms. We use neural networks to help train our AI. Neural Network is comparable to a human brain, in the way it would store and retrieve data as needed. We are using a specific policy called a CNN policy in this example, which is a really fast policy to process images, because our entire game is just a stack of frames which need to be processed. An important thing to notice here would be how many total steps we are going to use to train our AI. The best I could do on my machine was a 100,000 frames which is effectively how many frames our model will be looking at to get its data and use that to make decisions in the game further on. I have included images within the PPO section of the Approach subheading which shows us metric of the model. Ideally we would want the loss to go down and explained variance to go up as we are going through with the steps and frames which can be seen in the images, which is a positive indicator for the convergence of our model.

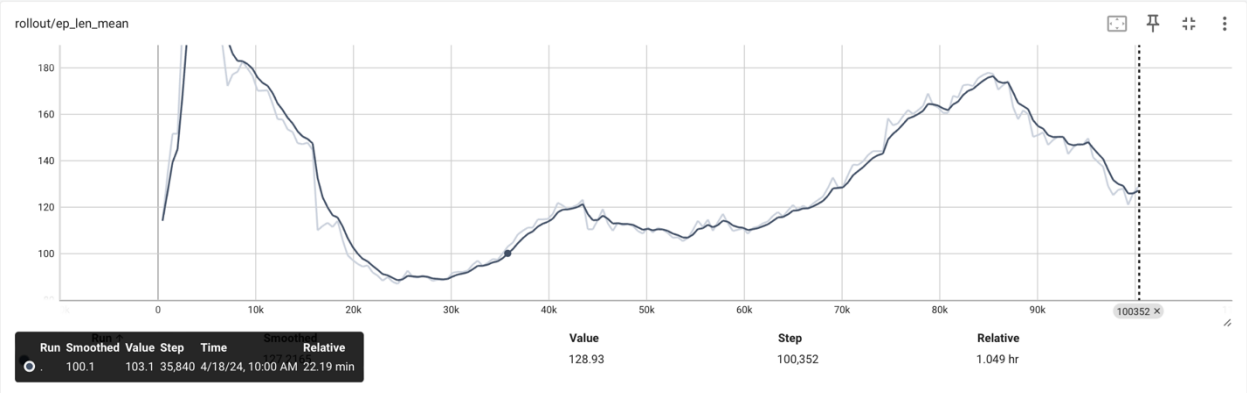


The graph above looks at the frames that are recorded per second relative to the time. We see that it falls between 26 and 28 frames per second and after 25,000 total steps, that number of frames processed per second becomes stable at 26 frames.



Now, this is one of the more interesting aspects. We see the episode reward mean for each of the time steps being calculated and we see that as number of timesteps increases the reward is going up and down. We start with really small rewards and then go on to higher rewards. This is based on number of factors, the reward function is designed by OpenAI, and this is an implementation of their reward function. I will include documentation for the reward function used below.

[3] "Part 3: Intro to Policy Optimization — Spinning Up documentation," *Openai.com*, 2018. https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html



This graph shows us the mean episode length for each of the timesteps. This shows how long it took Mario to die or to reach the goal because that is how we are defining each episode as and it calculates and spits out the mean of those values.

Another important evaluation is between the values show below.

rollout/	
ep_len_mean	27
ep_rew_mean	252
time/	
fps	24
iterations	553
time_elapsed	11634
total_timesteps	283136
train/	
approx_kl	0.0
clip_fraction	0
clip_range	0.2
entropy_loss	-5.05e-06
explained_variance	1
learning_rate	0.001
loss	0.0414
n_updates	5520
policy_gradient_loss	1.03e-07
value_loss	0.131

rollout/	
ep_len_mean	27
ep_rew_mean	252
time/	
fps	24
iterations	554
time_elapsed	11655
total_timesteps	283648
train/	
approx_kl	0.0
clip_fraction	0
clip_range	0.2
entropy_loss	-4.51e-06
explained_variance	1
learning_rate	0.001
loss	0.0231
n_updates	5530
policy_gradient_loss	1.96e-09
value_loss	0.0934

These are images of two consecutive iterations for the PPO algorithm. Ideally, we would want the loss to go down and the entropy loss to go up, and we see that this is the case which indicates that the model is converging.

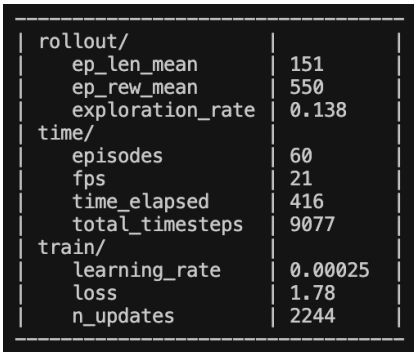
Now, onto the results. We have used the help of Nicholas Renotte's repository which is linked in the document elsewhere to achieve the task of using the PPO model, to train our Mario Agent. Nicholas has a total timestep of 4 million and it took multiple days for him to train his AI model to pass the level. We do not have access to computers with such capability, so we rolled

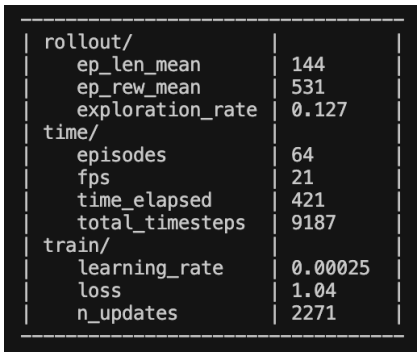
out at 100,000 total timesteps to train our agent and we have achieved some impressive results. Our Mario agent, although cannot pass the first level, can however show the progress he has made to learn and demonstrates what he has learned and implements it very well within the game. A demonstration video is provided which will show how we have made our agent learn and what progress he demonstrates.

Let's discuss the DQN Approach:

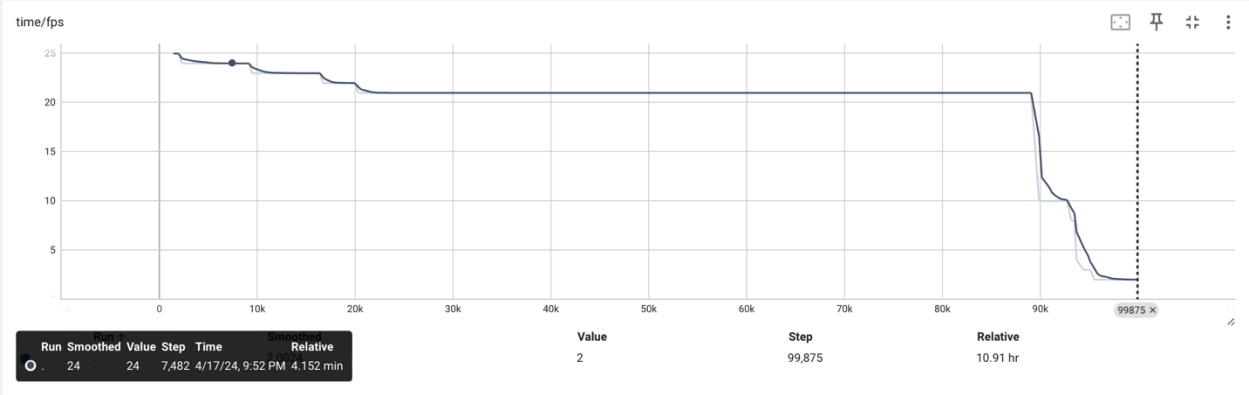
The approach for the implementation is very similar to the PPO algorithm and would just be redundant to include all the same information again. We are using all of the same libraries and packages as able expect we will be using the DQN algorithm instead of PPO algorithm to achieve a model that will try to complete the Mario level.

Let's talk statistics and result for this algorithm. I have included images that talk about different parameters for Mario for each of the timestamps, for consecutive iterations.

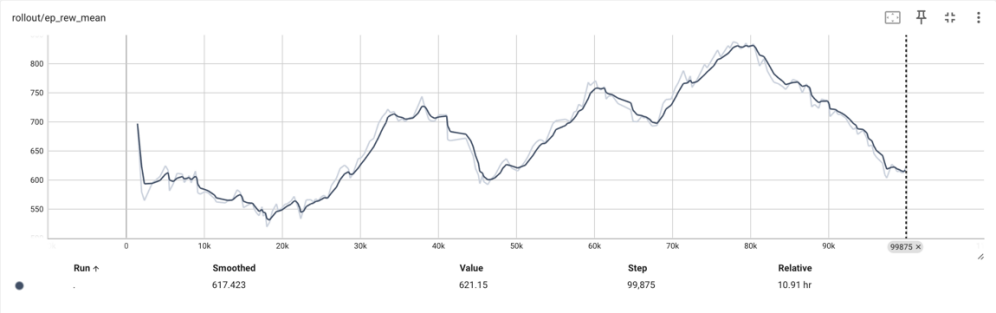




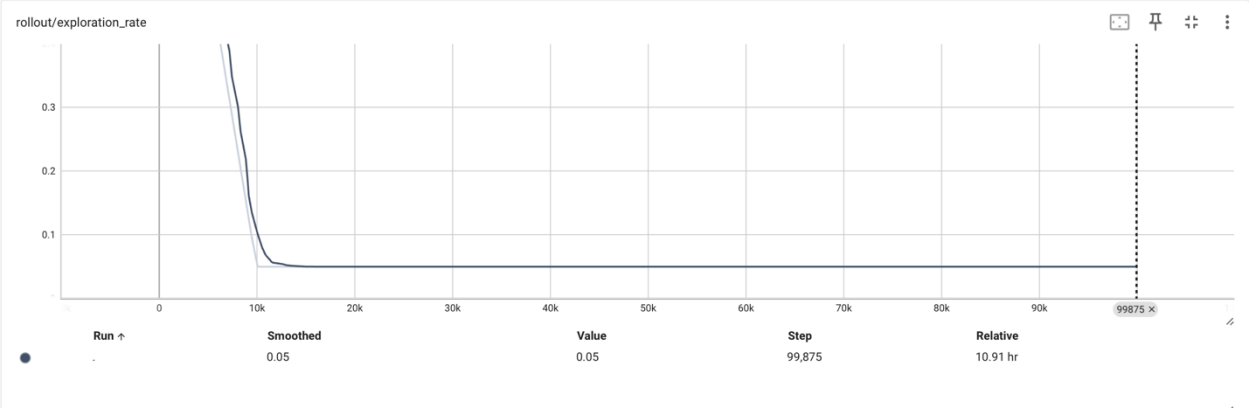
The metrics are going to be the same, we would ideally be looking at the loss and would want to see that to go down with each episode and we see that this is the case here as well. The loss parameter is going down, which is pointing towards the convergence of the model.



This is the graph that deals with the frame per second for each of the episodes until it reaches 100,000 total timesteps.



This graph deals with the episode reward mean which calculates the rewards for each of the timesteps. The reward function has a trend of increasing rewards which points towards positive learning rate. The reward function is increasing locally which points towards learning actions which result in higher rewards which point towards convergence again.



This graph points towards the exploration rate and we see that the graph becomes stable at 20,000 steps which points towards convergence. We must keep in mind of the results that we will be discussing later. Convergence does not mean we are looking at an ideal state for the agent and that we would be able to reach the final state, but it points to a stability in the state-action pair regarding decision.

We have followed Sourish's video example to complete the task and he runs his DQN algorithm for days before he is able to achieve convergence. He tries to work his algorithm in terms of iterations and runs it for 50,000 iterations before being able to converge. He runs his algorithm with an RTX graphics card which we do not have access to and do not have the computational power to achieve this task. I am rolling out the iterations after 100,000 timestamps because that is where my computer starts giving up on rewards and takes the decision to move only right after a while, and it appears that we achieve good results when we end execution at 100,000 steps.

For both the algorithms, we achieved a smaller convergence where the agent was able to pass a few obstacles and is able to decide whether to jump or not based on the obstacles present in the path. We are able to verify this in the demonstration video provided in the discussion section. Both the algorithms PPO and DQN are widely used for game training agents, but achieving true convergence is a very difficult task considering that every changing nature of the game and the dynamic environment and the huge action space which results in very complex computation. We were able to achieve satisfactory results with the amount of time we have trained our agent for both the algorithms, and the results point out to DQN being the better algorithm which has been inferred by our hypothesis and mathematical proofs. More discussion on this in the demonstration video.

Conclusion:

We will be discussing the results of running PPO and DQN algorithms. These are two algorithms that are widely used to train game agents to achieve the tasks to go to convergence and to finish the given level for a game. We are using these algorithms to compare and contrast the difference between these algorithms and to see how they function, and which one is better suited and tailored to complete the Mario game. Mario game utilized both the algorithms fashionably, but it seems like DQN is more prone to achieving convergence faster and in a more efficient way. Based on what we have observered, DQN is able to perform better and more logically with the amount of training it has received so far.

We have learned numerous new concepts about the Mario agent and reinforcement learning in general. I had an idea about how difficult and time consuming it is to train a model and to achieve convergence, but I had not imagined the daunting task of implementing it and failing numerous times. Although we were able to get Mario to understand some of the dynamics of the game and make him counter some of the difficulties, it has been almost impossible to run the sheer amount of iterations and time steps to achieve convergence in this game. We were able to reach a point where we were comfortably sitting at 100,000 iterations to run the game, but we were not able to let the agent complete the level.

The sheer computational capabilities required to achieve this is a lot, and we do not have access to such computational power. We ran the training on our laptops with 8 gigs of RAM, and that has not been enough at all, for what we were trying to achieve.

We have learned numerous new concepts of frame stacking, gray scaling and using stable baselines, a very popular reinforcement learning package that deals with many algorithms that are used to train agents. We have implemented these concepts in our project, and we were able to

achieve satisfactory results for all the problems we tried to tackle. Super Mario has a very dynamic environment, and the action space is huge as well, so to train an agent to tackle a level is a daunting task, but nevertheless we were able to get close to the goal. The agent still is seen making sensible judgments and is able to perform many actions and also combine set of actions to get as close to the goal state as possible.

References:

[1] Y. Liao, K. Yi, and Z. Yang, "CS229 Final Report Reinforcement Learning to Play Mario." Available: https://cs229.stanford.edu/proj2012/LiaoYiYang-RLtoPlayMario.pdf

[2] S. Karakovskiy and J. Togelius, "The Mario AI Benchmark and Competitions," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 55–67, Mar. 2012, doi: https://doi.org/10.1109/tciaig.2012.2188528.

[3] "Build an Mario AI Model with Python | Gaming Reinforcement Learning," *www.youtube.com*. https://www.youtube.com/watch?v=2eeYqJ0uBKE&list=PLFNrAHO8tpSyv64czmtRuLwd9hrNCuxPp&index=2 (accessed Apr. 17, 2024).

[4] N. Renotte, "nicknochnack/MarioRL," *GitHub*, Oct. 02, 2023. https://github.com/nicknochnack/MarioRL

[5] "Train AI to Beat Super Mario Bros! || Reinforcement Learning Completely from Scratch," *www.youtube.com*. https://www.youtube.com/watch?v=_gmQZToTMac&list=PLFNrAHO8tpSyv64czmtRuLwd9hrNCuxPp&index=4 (accessed Apr. 17, 2024).

[6] "Super-Mario-Bros-RL/README.md at main · Sourish07/Super-Mario-Bros-RL," *GitHub*. https://github.com/Sourish07/Super-Mario-Bros-RL/blob/main/README.md (accessed Apr. 17, 2024).

[7] S. Padhye, "Playing Super Mario Bros. with Reinforcement Learning," *Medium*, Nov. 27, 2022. https://sohum-padhye.medium.com/playing-super-mario-bros-with-reinforcement-learning-81ee0c235372 (accessed Apr. 17, 2024).

[8] D. P. Teh, "Playing Super Mario with PPO Reinforcement Learning Agent," *Medium*, Feb. 16, 2024. https://medium.com/@darioprawarateh/playing-super-mario-with-ppo-reinforcement-learning-agent-e9f61fd04b32

[9] Y. Feng, "yfeng997/MadMario," *GitHub*, Mar. 26, 2024. https://github.com/yfeng997/MadMario (accessed Apr. 17, 2024).

[10] "PyTorch," *www.pytorch.org*. https://pytorch.org/get-started/locally/

[11] "PPO — Stable Baselines3 1.4.1a3 documentation," *stable-baselines3.readthedocs.io*. https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

[12] "Part 3: Intro to Policy Optimization — Spinning Up documentation," *Openai.com*, 2018. https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html

Appendix

Aryan Miriyala:

Contributions:

I have worked on the documentation, bug fixes, testing and training of the model and to have everything organized so we could get the project completed in a timely manner. I oversaw finishing the Project Report which took a large chunk of time. I had to find all the resources and information regarding the training agent and the reinforcement

learning algorithms used and I also oversaw presenting the overview and demonstration for our project. I prepared the slides and the content and researched enough to have meaningful content in the slides and in the project. I prepared the documentation so that the code is completely and properly executable and running. I organized the timeline and prepared the project proposal for the group. I made the required changes to the existing code created by Cedric and the external sources and made sure everything is functioning.

Lessons Learned:

I learned a lot of new concepts that deal with reinforcement learning and artificial intelligence. I learned about image processing policies – especially the cnn policy which deals extensively with processing of images as the entire game is a stack of frames, and we had to train our agents based on these frames. I learned about wrappers in terms of frames and the functions that are useful to break the game down into wrappers. We utilized a lot of concepts regarding game training, policy refinement, Deep Neural Networks, and Policy Optimization. It was very interesting to apply the concepts we learned in our class to real world project and see them run first hand.

Cedric Dortch:

Contributions:

For this project, I was mainly in charge of setting up/preprocessing the environment to make sure that everything was working and ready for testing. This required a lot of learning and debugging the code. Since some of our sources are outdated a lot of the libraries would not work properly with the code that was written, so I had to find loopholes in order to get them to work.

I was also responsible for the documentation for running the code. On GitLab I documented how to set up the environment, install requirements and run the code effectively in the "How to Run the Code" file.

Lessons Learned:

I previously had no knowledge on the PPO or DQN artificial intelligence algorithms, so I needed to do a lot of research on the algorithms and how to run them effectively for our Mario agent. I learned about the role of artificial intelligence in training models to learn from their environment. All the training of our Mario agent was through image processing. We would capture an image of our environment then we would use AI to learn from that image with the CNN Policy. Image processing played a major role in our project, and I think it is one of the valuable lessons that I learned while completing this project.