

Computer Games Development

Project Report

Year IV

JoshuaDunne

C00241588

14/11/2021

**Contents**

[**Acknowledgements**](#_heading=h.z2h0j2rx0vio)1

[**Project Abstract**](#_heading=h.6nqy007lspi5)3

[**Project Introduction and/or Research Question**](#_heading=h.580uk0lc0vh)4

[**Literature Review**](#_heading=h.kzk93a6s9lmh)5

[**Evaluation and Discussion**](#_heading=h.lpzwaa25p6m3)6

[**Conclusions**](#_heading=h.1ef4cba684ct)14

[**References**](#_heading=h.u2v46im2tre5)14

[**Appendices**](#_heading=h.1t3h5sf)16

# Acknowledgements

I would like to thank the following people who assisted in completing this project including;

My project supervisor, Oisin Cawley, for helping to keep me on track with my progress, and giving me advice on what way to go with the project overall.

Nicholas Renotte for creating an incredibly helpful video that helps understand the basics of Reinforcement Learning and how to apply it for OpenAI Gym. [1]

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave and Anssi Kanervisto for creating and maintaining Stable Baselines 3, which helped greatly with policy algorithms to help make the process of creating models and using them on environments a lot easier. [2]

The many people behind creating Gym, which is part of Open AI, making it easier to try out environments other people made to help further my understanding of how to make one myself. [3]

# 

# Project Abstract

For Agent Based Modelling, Reinforcement Learning is used to teach a Model how to approach a solution for an Environment. For example, driving around a track or attempting to drive up a steep hill.

Typically these Environments would have a set of straightforward actions and rules, allowing the Agent in the Environment to efficiently reach the end goal of completing the task given to it, accumulating rewards effectively.

These Agents use algorithms, a set list of actions and gain rewards in order to determine what their end goal is. I wish to look into adding randomness to see how it affects their overall approach to learning the best way to maximize reward gain.

By introducing randomness into these otherwise static environments, I hope to gain new knowledge on why exactly randomness isn’t used in most Environments as a standard, and hope to see if I can find a reason as to why or why not randomness should be used in all Environments

However, randomness is not a new concept to Reinforcement Learning, as randomness can be used in some scenarios to make Agents try alternative paths. For example, imagine an environment where the Agent can pick two doors. The Left Door gives +1 reward, and the right door only has a 10% chance to give +100 reward, otherwise it gives 0 reward. If the Agent picks the Right Door and gets nothing, then picks the Left Door, it will assume that picking the Left Door is always the correct choice, despite the fact the Agent could be getting +100 reward from picking the Right Door.

I will be approaching this thought process using already created Open AI Gym Environments as a base line, introducing randomness into each and comparing it against Agents that run through the Environment without randomness, in an effort to further understand exactly why randomness isn’t standard in all Environments.

# 

# Project Introduction and/or Research Question

The main questions I wish to explore are why non-deterministic elements (aka randomness) are not an inherent property of all Environments and whether or not non-deterministic elements are a good thing in Reinforcement Learning.

Introducing non-deterministic elements into Reinforcement Learning is important, because it’s almost impossible to determine certain outcomes or placement of objects during a real time scenario, and forcing an Agent into a scenario where everything goes perfect every time leads to narrowing on the Agent properly learning how to deal with a scenario. For example, if you were teaching an Agent how to make a turn on a road, your Environment wouldn’t be able to account for all real life scenarios that could arise, such as animals being in the way, other cars or possibly people trying to cross the road at the same time.

For this project, I will be using Model-Free Agents, meaning they are not based on a pre-existing model and learn their Environments from scratch.

Environments are play areas that Agents interact with in order to gain rewards. If an Agent picks an action that furthers the end goal, they gain rewards, otherwise they will lose rewards.

Rewards in this scenario are given to the Agent when it makes a correct decision, or picks an action that moves them further towards the end goal. For example, in the hill climbing Environment, the Agent gains a reward the higher it climbs up the hill, and loses reward for each second that passes, incentivising getting to the top of the hill as soon as possible.

In short, an Agent picks an Action, which is given to the Environment. The Environment then processes this information, and gives back a reward to the Agent and a new state for the Agent. This allows the Agent to learn as it interacts with the Environment.

By introducing non-deterministic elements, I hope to gain more of an understanding on why they aren’t the usual standard in most Environments, and if non-deterministic elements should even be considered for Reinforcement Learning.

By the end of this project, I hope to have introduced non-deterministic elements into two pre-built Environments, and to create my own Environment where I can try other forms of randomness at my own pace.

# 

# Literature Review

Reinforcement Learning [13] is used to train an agent to learn from interacting with an environment in order to reach a goal. It is a very good approach to solving decision-making problems. Agents discover which actions give the greatest reward by trying out actions on a certain state, and learning how good that action was by judging how good the state is that they are now in from that action. Reinforcement learning wants to maximize the overall reward.

Rewards are given by the environment based on the agent's actions and the state given back. Since states / actions with the highest values end up giving the best reward ultimately, we want to focus on the value of the state / action when making decisions.

There are two different types of models that can represent an agent, Model Based and Model Free. For this project, we will be focusing on the Model Free aspect, as we want to have the agent fully learn based on these actions, states and rewards to end with a model that can solve the environment entirely through reinforcement learning, without having to rely on a pre-existing model. This model will be based on Proximal Policy Optimization ([7], [14]), using Stable Baselines 3’s implementation [15].

PPO models are typically used in static environments, whereas my intent is to attempt to push them through dynamic environments, where non-deterministic elements exist. This can cause the models to enter states they are not expecting, which is where my research comes in. I wish to investigate what I can potentially do to try to mitigate the possibility of an agent getting stuck in a state it is not expecting.

My proposed solution is to catch the actions the model is picking, and see if they end up repeating. If the repeating actions are opposites, the model will be told to learn the current board state without the board changing, in hopes that the agent will then learn how best to approach the board state it was not expecting.

# 

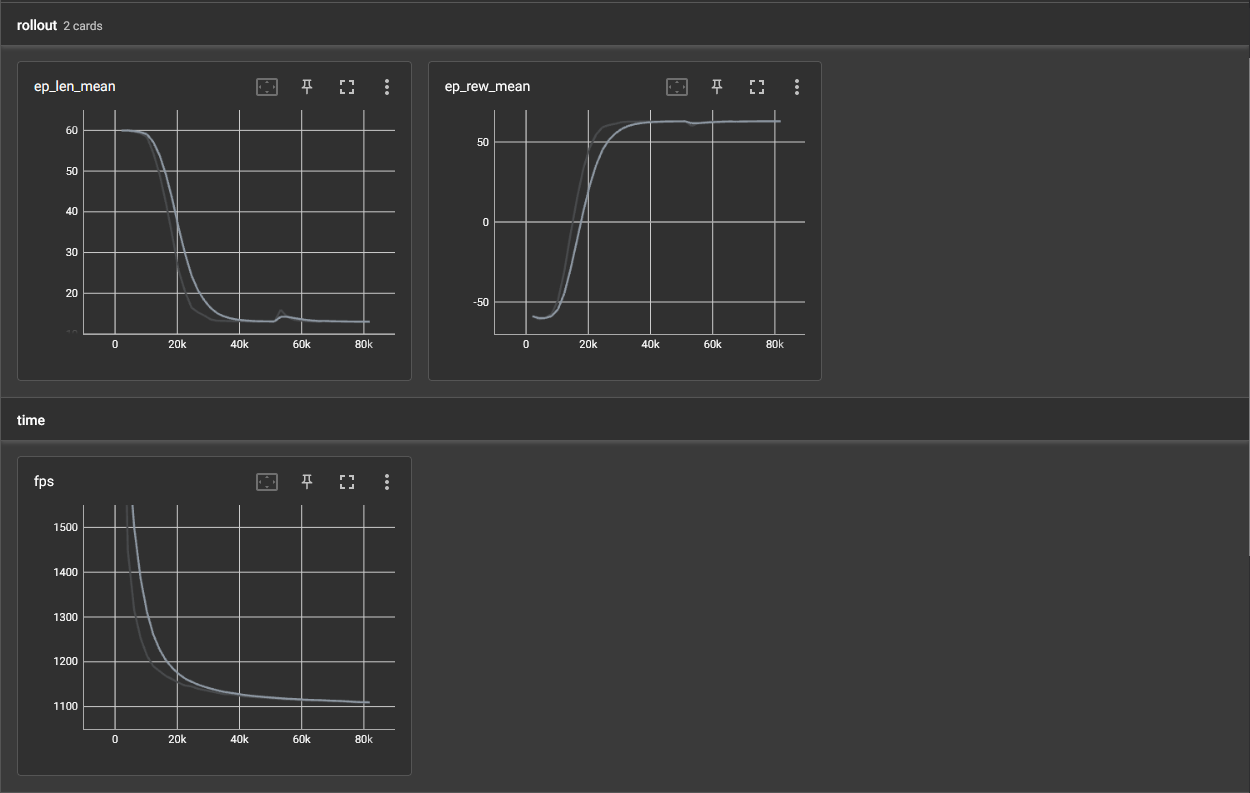
# Evaluation and Discussion

The idea I had came up with was to create a simple environment which would have a board, and a position for the Player on the board. With this set up, I then implemented board pieces that would push the Player either forward or backwards. Using this, I could randomize the board and see how a model would attempt to learn an environment which was non-deterministic, or a dynamic environment.

After about 80,000 timesteps, a model trained on a pre-set version of the environment, one where no board pieces are randomized, was able to easily move through the maze with minimal mistakes, more often than not taking the correct path.

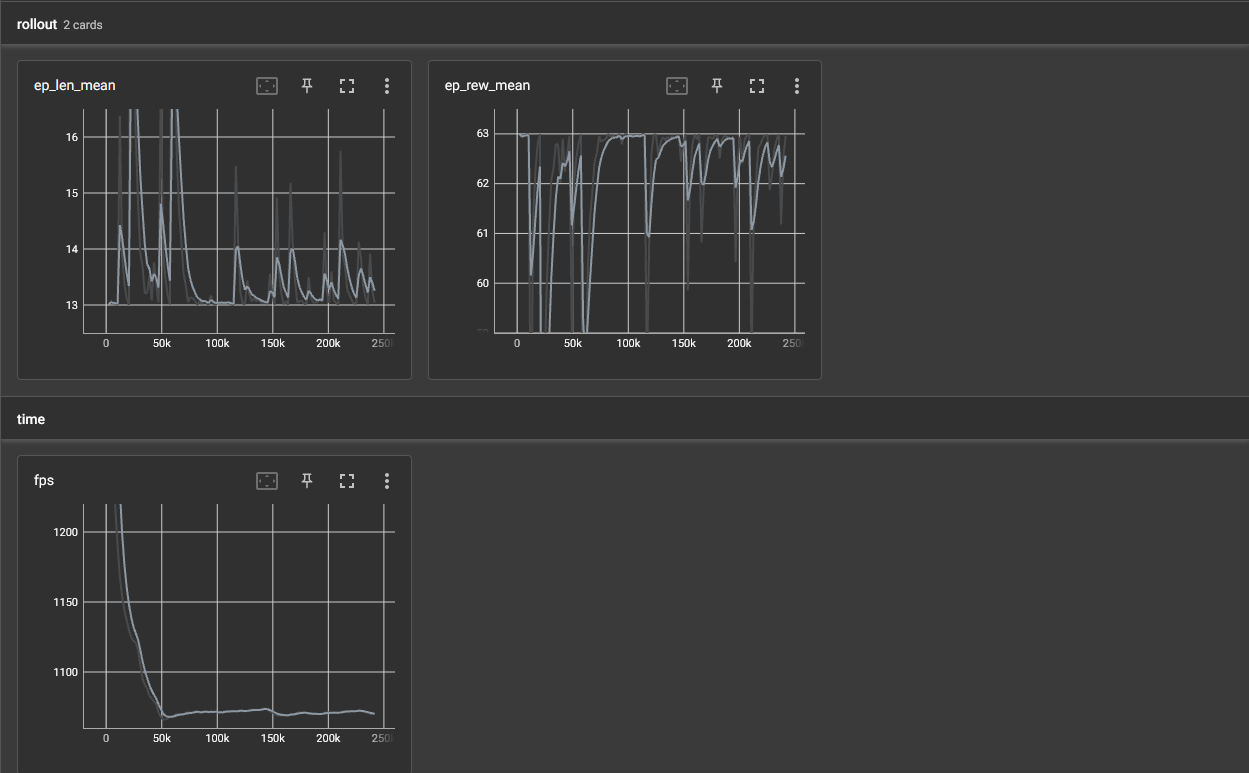
With a further 240,000 timesteps, totaling to 320,000 timesteps, the model became even more efficient in the pre-set environment, having a perfect success rate of about 99%.  
  
In the following screenshots, we will look at the average length and reward that each model got after their specified timestep length.

**No Randomness 80k:**



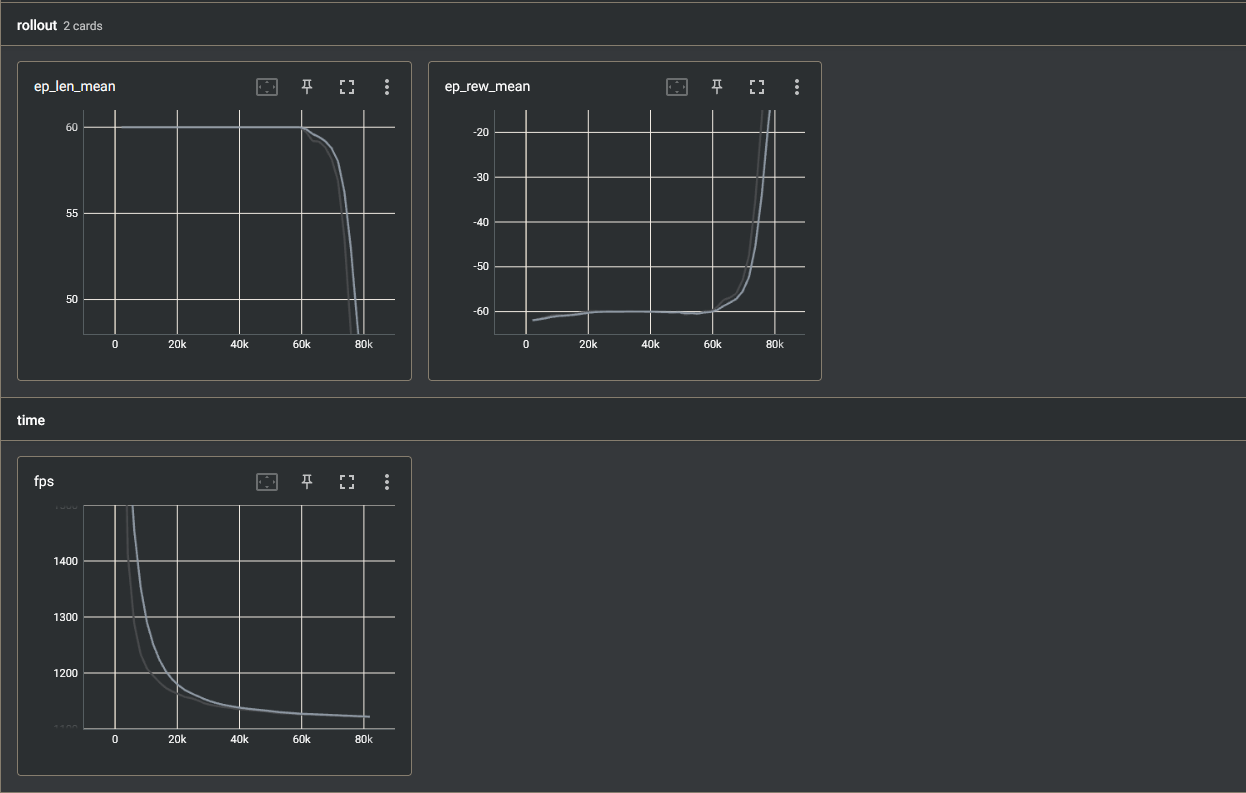
***Averages of Length of Episodes & Rewards gained within the Episodes for 80k***

**No Randomness 320k:**

***Averages of Length of Episodes & Rewards gained within the Episodes for 320k***

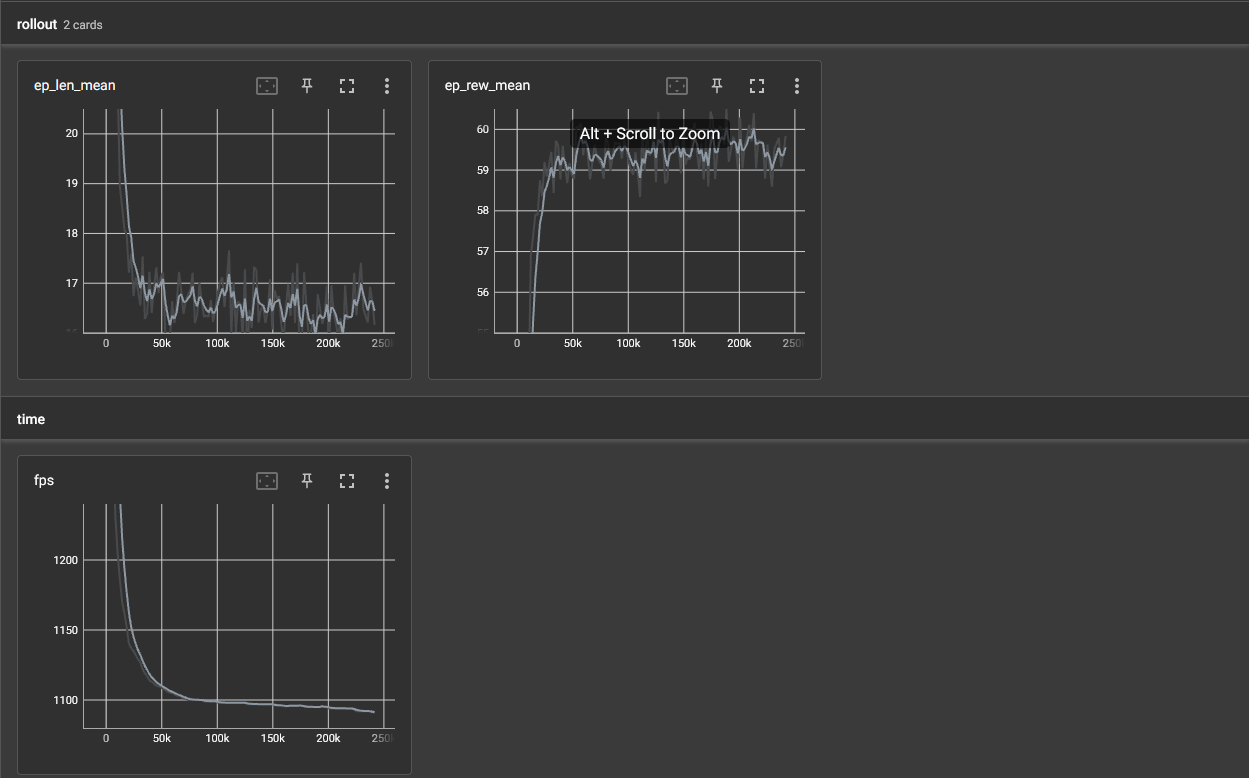
The 320k timesteps model seems to end up with a more erratic length and reward mean, however, I believe this is due to Exploration vs Exploitation. While the model can correctly take the path each time now that it has figured it out, the algorithm will allow the model to pick actions that may not have been explored before, in order to gain more knowledge on what actions are the best for each observation state.  
Without this technique, the model would end up taking the same path each time, and while it is correct, if the environment were to suddenly change, the model would get stuck very fast, which is what I observed when I tried to use my No Random model in my environment with non-deterministic elements turned on.

**Randomness 80k:**



***Averages of Length of Episodes & Rewards gained within the Episodes for 80k***

**Randomness 320k:**

***Averages of Length of Episodes & Rewards gained within the Episodes for 320k***

As it can be observed, the 80k model struggles to get a good average length and reward, which is to be expected, as the model would not be able to generate enough data based on random variables to become good enough to determine what to do when the blue space is not where it is expected to be. This causes the model to pick the same opposite actions repeatedly in an attempt to move onto the space where it expects to be pushed.  
  
Another interesting observation can be made when it comes to training a model on the environment with non-deterministic elements active.  
With this option enabled, the blue spaces that push the Player forward are randomly placed along a column, meaning they will be in different positions each time.

A consequence of this ends up being that the average length and reward gained seems to fluctuate extremely in comparison to the previous 80k and 320k model logs with randomness disabled.

**Project Milestones**

In terms of milestones, I have managed to keep on schedule for them. For each week, I showed my project supervisor what work I had done, and decided on what to do next, as well as discussed issues I had with the project to them  
Each meeting I had with the project supervisor was where I would determine what to do for the next meeting. I kept a log of each week, making sure that we both knew what I was doing from week to week.

While this type of planning might not be the usual calendar based planning, I felt it worked for a more proactive project development, rather than a reactive project development, where I would have to adjust a pre-scheduled plan as I worked on the project.

**Major Technical Achievements**

As a result of doing my project the way I have, I have come out of the other side with my own version of a grid-based environment which allows for randomization to inspect the effects of running models against a dynamic environment.  
  
If anyone else past me decides to look into the same field as I, they could easily download the code I have created and run the environment themselves. On top of this, I tried to make the code as easy to understand as possible, making it simple for other people to edit the environment as they so fit, so they could include other space types on the board / grid, as well as remove existing space types.  
In my implementation, the start and goal are always stationary, but other people using my environment can easily move those starting and ending positions on the board easily.

**Project Review**

**What went right?**

Overall, the project finished with a pretty solid implementation of a custom environment that allowed for easy switching of non-deterministic elements to change it from a static environment to a dynamic environment. With the full implementation, it made it a lot easier to train models under different circumstances, and compare their data, as well as view their behaviours when both models interacted with the other’s environment.  
  
On top of this, my current working implementation of dynamic re-training of the model is simple, and should be easy to tweak and implement into any other environment that does not feature a complex action space.

**What went wrong?**

With this project, the majority of time was spent looking mostly into how to get environments running, or looking into injecting randomness into those existing environments. I had originally intended to do this with two different environments, Self-Driving and Cartpole. However, after trying it with Cartpole, I ended up realizing it was a fruitless effort, as those environments would already calculate nearly every possibility themselves, and were able to account for the sudden randomness, since the randomness would simply just move the model into a different state it already knew how to deal with.  
  
For example, in Cartpole, I randomly increased the pole’s rotation speed, making the pole suddenly move faster in one direction. While I had presumed the model would be unsure what to do in the situation where it went from a normal state to this abnormal state, it was quite the opposite. The model already knew what to do to stop itself from losing in the environment, as it already went through each pole rotation speed state during training.

**What (if anything) is still outstanding/missing (i.e., still left to do)?**

While there is nothing specifically left with the actual final implementation of my project, if I had more time to work on the project, I would’ve liked to continue looking into the process of re-training. With my current implementation, the loop that uses the model has to check itself to see if the model has been caught in a loop of similar decisions. However, I would much rather try to do this during either the initial training, or through the model itself when the model is being applied.

**If starting again, how would you approach this project differently?**

I would like to try writing the code for the model myself, so I had more freedom with what the model can do, and more access to the internals, as well as having a better understanding overall on how models work in regards to Exploration vs Exploitation in a dynamic environment.

**What advice would you have for someone attempting a similar project in the future?**

I would definitely advise anyone else attempting similar to try the above mentioned approach. While it would be more difficult, it wouldn’t be too much hassle to have a look at the repositories for Open AI Gym [3] and Stable Baselines 3 [2].

**Were your technology choices the right or wrong ones?   
If you chose the wrong technology, provide justifications for why you think this.**

The technology choices were the correct ones. Stable Baselines 3 and Open AI Gym give direct and easy access to reinforcement learning, making it so anyone new to the subject can easily pick it up and understand how it works.

**What were the implications of your technology choices?**

By choosing to use libraries rather than making the code myself, I saved time on writing the code myself, but lost out on actual implementation understanding. While I understand how reinforcement learning works on paper, I wouldn’t have any ideas on how to actually write the code myself to achieve what the people behind Stable Baselines 3 and Open AI Gym have done.

# 

# Conclusions

*summarise your work and findings.*

Overall, I believe my implementation of checking for duplicate actions and re-training does help with the issue of agents getting stuck in a state it was not expecting, causing it to repeat actions until it picks an action that moves it off the unknown state. However, it is not perfect, as the fix is not universal. For example, this same solution may not work for an agent learning to play Go, which features technically dynamic gameplay as the agent may have to play against a player who can place their pieces anywhere.

Future Work

The next steps to potentially try in this subject are definitely to look further into how to automate the process of having the environment realize that the model is taking the same actions, or looking into how you could use the model to dictate if it thinks it should be in a different state.

For my implementation, I simply just use actions given by the model, but it might be possible with more time to check the internals of the model to determine if the model expects to be where it is currently. If it is somewhere it’s not expecting, you could use that factor to trigger re-training and train for this unchecked area.  
  
If I were given more time, I would’ve liked to try to create a more universal solution that could be applied to any dynamic environment, as my current solution only works on my environment.

# References

*[1] Renotte, Nicholas. “Reinforcement Learning in 3 Hours | Full Course Using Python.” YouTube, 6 June 2021,* [*https://www.youtube.com/watch?v=Mut\_u40Sqz4*](https://www.youtube.com/watch?v=Mut_u40Sqz4)*.*

*[2] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, & Noah Dormann (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. Journal of Machine Learning Research, 22(268), 1-8.*

*[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, & Wojciech Zaremba. (2016). OpenAI Gym.*

*[4] Tewari, U. (2020, April 14). Which reinforcement learning-RL algorithm to use where, when and in what scenario? Medium.Datadriveninvestor.Com. Retrieved January 27, 2022, from* [*https://medium.datadriveninvestor.com/which-reinforcement-learning-rl-algorithm-to-use-where-when-and-in-what-scenario-e3e7617fb0b1*](https://medium.datadriveninvestor.com/which-reinforcement-learning-rl-algorithm-to-use-where-when-and-in-what-scenario-e3e7617fb0b1)

*[5] Base RL Class — Stable Baselines3 1.5.1a4 documentation. (2020, June 9). Stable-Baselines3.Readthedocs.Io. Retrieved January 27, 2022, from* [*https://stable-baselines3.readthedocs.io/en/master/modules/base.html*](https://stable-baselines3.readthedocs.io/en/master/modules/base.html)

*[6] Moni, R. (2021, December 7). Reinforcement Learning algorithms — an intuitive overview. Medium. Retrieved January 27, 2022, from* [*https://smartlabai.medium.com/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc*](https://smartlabai.medium.com/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc)

*[7] Simonini, T. (2019, February 5). Proximal Policy Optimization (PPO) with Sonic the Hedgehog 2 and 3. Medium. Retrieved January 27, 2022, from* [*https://towardsdatascience.com/proximal-policy-optimization-ppo-with-sonic-the-hedgehog-2-and-3-c9c21dbed5e*](https://towardsdatascience.com/proximal-policy-optimization-ppo-with-sonic-the-hedgehog-2-and-3-c9c21dbed5e)

*[8] freeCodeCamp.org. (2018, March 29). An introduction to Deep Q-Learning: let’s play Doom. Retrieved January 27, 2022, from* [*https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/*](https://www.freecodecamp.org/news/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8/)

*[9] Reinforcement Learning Tips and Tricks — Stable Baselines 2.10.2 documentation. (n.d.). Stable-Baselines.Readthedocs.Io. Retrieved January 27, 2022, from* [*https://stable-baselines.readthedocs.io/en/master/guide/rl\_tips.html#which-algorithm-should-i-use*](https://stable-baselines.readthedocs.io/en/master/guide/rl_tips.html#which-algorithm-should-i-use)

*[10] Wikipedia contributors. (2022, April 12). Multilayer perceptron. Wikipedia. Retrieved March 24, 2022, from* [*https://en.wikipedia.org/wiki/Multilayer\_perceptron*](https://en.wikipedia.org/wiki/Multilayer_perceptron)

*[11] An Overview on Multilayer Perceptron (MLP). (2022, February 21). Simplilearn.Com. Retrieved March 24, 2022, from* [*https://www.simplilearn.com/tutorials/deep-learning-tutorial/multilayer-perceptron*](https://www.simplilearn.com/tutorials/deep-learning-tutorial/multilayer-perceptron)

*[12] Miyoung Han. Reinforcement Learning Approaches in Dynamic Environments. Databases [cs.DB]. Télécom ParisTech, 2018. English. tel-01891805*

*[13] R. S. Sutton and A. G. Barto. Introduction to Reinforcement Learning. MIT Press, 1998.*

*[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv preprint arXiv:1707.06347, 2017.*

*[15] PPO — Stable Baselines3 1.5.1a4 documentation. (n.d.). Stable Baslines 3. Retrieved January 27, 2022, from https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html*

# Appendices

*Replace this text with Appendices.*

*This might include ethics application and other relevant material e.g. copy of any questionnaires used.*