# Indian Institute of Technology Gandhinagar


---

## Reinforcement Learning Project Report

---

# ME 499: "Exploring the Efficacy of PPO Reinforcement Learning Algorithm in Navigation Tasks with Obstacle Avoidance"


### AUTHORS

*Ashutosh Goyal 20110029*

*Rajesh Kumar 20110161*

### Under The Guidance Of:

*Prof. Madhu Vadali*

*Suraj Borate 20310037*

## Contents:

## Abstract:-

The project aims to train an agent to navigate through a simulated environment toward a designated target while avoiding obstacles. The project first experimented with a discrete observation space and action space, with the agent's position, obstacles, and target location being discrete. The agent's actions were also limited to discrete movements, specifically up, down, left, and right by one step. Through experimentation, it was observed that training the agent for 4-5 lakh steps was sufficient to achieve complete training for the navigation task while avoiding obstacles.

After success in the discrete environment, the project moved to a continuous environment, where the observation space was continuous, meaning that the position of the agent, obstacles, and target could be anywhere within the specified environment. Additionally, the actions available to the agent were also continuous. However, the transition to a continuous environment required a different approach to training the agent, and the project struggled to achieve consistent and reliable training of the agent in a continuous environment.

To address the challenge of training the agent in a continuous environment, the project utilized the properties of the Pygame environment by using the rect.right, rect.left, rect.up, and rect.down functions of Pygame to obtain the boundaries of the agent. This allowed for the successful training of the agent in a continuous environment. However, the agent's movement remained fuzzy, so the project modified the action space to be based on acceleration, resulting in smoother agent movement. To further improve the agent's performance, the project needed to tune the hyper-parameters of the reinforcement learning algorithm.


## Introduction:-

Reinforcement learning is a subfield of machine learning that involves training an agent to make decisions based on its interaction with an environment. In this project, we aim to train an agent using reinforcement learning to reach a target position while avoiding obstacles. The task involves designing an environment that includes the agent, obstacles, and a target location. The agent receives rewards for successfully reaching the target and penalties for colliding with the obstacles.

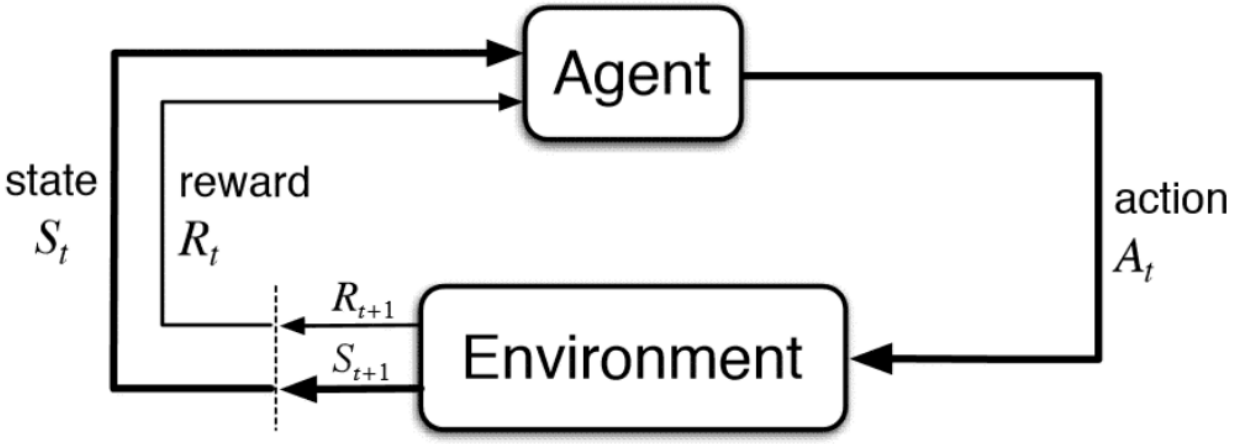



*Image Source*

Reinforcement learning involves the agent engaging with the environment through the observation of its states. Using these states as input, the agent chooses an action to execute, which results in a transition to a new state in the environment. This process repeats iteratively as the agent receives feedback in the form of rewards, allowing it to learn optimal behavior.

The goal of this project is to explore the effectiveness of reinforcement learning techniques in solving navigation tasks. Specifically, we aim to investigate how well the agent can learn to navigate through a complex environment while avoiding obstacles. We will be using the PPO policy algorithm, a widely-used reinforcement learning technique, to train the agent.

The project will involve several steps, including designing the environment, implementing the PPO policy algorithm, and evaluating the performance of the trained agent. We will also explore different parameters and hyperparameters of the algorithm to determine their effect on the agent's performance.

Overall, this project aims to demonstrate the potential of reinforcement learning in solving complex navigation tasks. The results of this project could have applications in fields such as robotics, autonomous vehicles, and gaming.

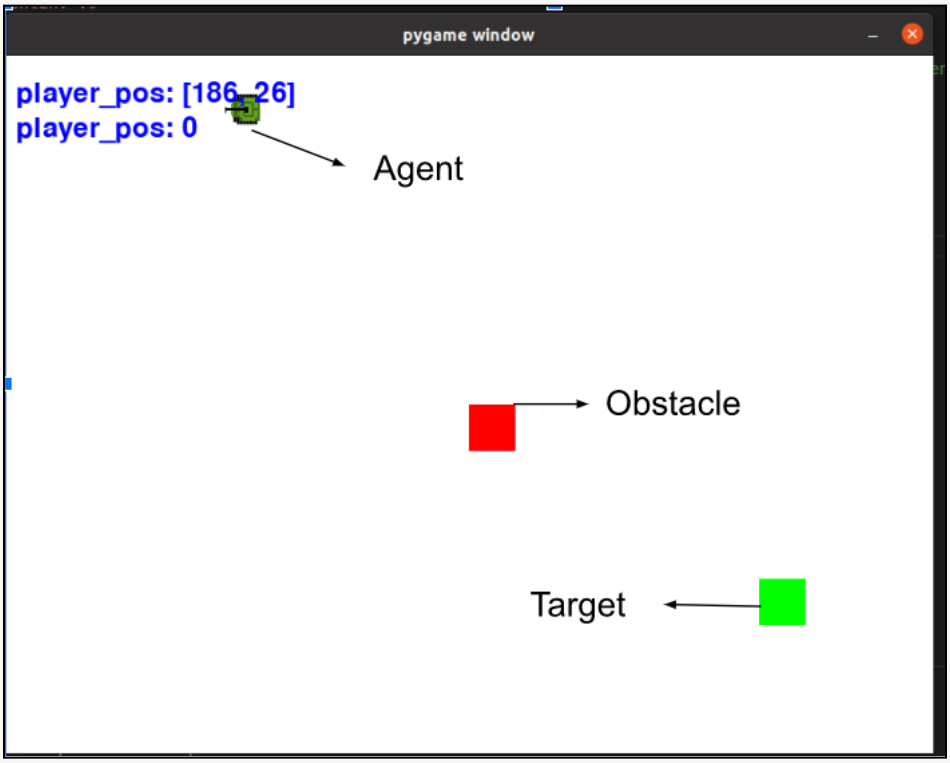
## Model Development:-

For our project, we have utilized the gym environment along with the PPO policy provided by the stablebaselines3 library to train our agents. Additionally, we have incorporated the pygame library to create a display window. We have selected the PPO policy as it has shown superior performance in the given scenarios. Our project involves working in both discrete and continuous (Box) domains, and after careful consideration, we have determined that PPO is the most suitable algorithm for our task.

| Space | Action | Observation |
|---|---|---|
| Discrete | ✔ | ✔ |
| Box | ✔ | ✔ |
| MultiDiscrete | ✔ | ✔ |
| MultiBinary | ✔ | ✔ |
| Dict | ✘ | ✔ |

*Image Source*

Now, the two main components while training the agent are creating the agent environment and the rewards distribution.




1. **Agent Environment:-**
    - ***For Discrete Action space and Discrete Observation Space***

- The code defines a custom environment called "**AgentEnv**" for an obstacle avoidance task where an agent is required to reach the target position while avoiding obstacles. The environment is based on the OpenAI gym library and uses Pygame for rendering.

- The environment is designed with a grid layout of size 40 units, where the agent and obstacles of the same size are placed. The agent can take discrete actions such as moving up, down, left, or right to navigate on the grids. There are three different environments in which the observation space varies depending on the motion of the obstacle and target.

- In the first environment, the obstacle and target positions are fixed, and the observation space is discrete and only consists of the current position of the agent on the grid. The agent does not know the position of the obstacle unless it receives a high negative reward, and it indirectly determines the position of the target based on the reward. However, if the obstacle position is changed after training the agent, it performs poorly as the trained agent does not have information about the current position of the obstacle.
- In the second environment, the obstacle position is randomized, and the target position is fixed. The observation space consists of the current position of the agent and the obstacle on the grid. The agent is aware of the obstacle's position and takes action to avoid it and reach the target. In this case, the agent can perform better as it has information about the obstacle's position in the observation space.

- In the third environment, both the obstacle and target positions are randomized, and the observation space consists of the current position of the agent, obstacle, and target on the grid. The agent has information about the obstacle and target positions and can navigate the grid accordingly. It can avoid obstacles even after training in a different environment where the positions of the obstacle and target are different.

- The environment is initialized in the constructor method using Pygame, where the screen and grid dimensions are set. The action space and observation space are defined as discrete spaces, and the images for the agent, target, and obstacle are loaded and scaled to the grid size.

- The **step method** is used to update the environment state based on the agent's actions. It takes action as input and updates the agent's position,

calculates the reward based on the distance between the agent's position and the target position, and determines whether the episode ends. The state returned by this method varies based on the environment: the current position of the agent in the first environment, the current position of the agent and obstacle in the second environment, and the current position of the agent, obstacle, and target in the third environment.

- ➢ The **reset method** is used to reset the environment state and return the initial state of the environment. The initial state varies based on the environment: the initial position of the agent in the first environment, the initial position of the agent and a randomized position of the obstacle in the second environment, and the initial position of the agent, randomized position of the obstacle, and target in the third environment.

- ➢ The **render method** is used to render the current state of the environment. It displays the current position of the agent, the target position, the obstacle, and the current score. The method takes a score parameter, which is the current total reward of the agent in the current episode.

The code is written in Python and uses Pygame for rendering and OpenAI gym for defining the environment. The code is written in an object-oriented style and follows the standard conventions for defining an OpenAI gym environment. The environment is designed to be used for training and evaluating reinforcement learning algorithms that require a discrete action and observation space.

- ***For continuous Action space and continuous Observation Space***
  - ➢ In this task, we have extended our previous work of creating a custom environment with a discrete action space and continuous observation space to a new environment with a continuous action space and continuous observation space. We modified variables such as *action space, observation space, state, rewards, velocity, acceleration, and methods* of the customs environment to fit the requirements of the task.

  - ➢ The continuous observation space includes the positions of the agent, obstacle, and target. In contrast, the continuous action space comprises the acceleration of the agent.

  - ➢ In the step function, we updated the agent's velocity and position based on the action received during training. We checked if the agent collided with the obstacle or reached the target and rewarded or penalized it

accordingly. If the agent reached the target, it received a large positive reward, and the episode ended. However, if the agent collided with the obstacle, it received a large negative reward, and the episode ended. Otherwise, the agent was rewarded based on its distance from the target. The step function returned the new state, the reward obtained from the previous action, and a boolean value indicating whether the episode had ended. If the episode had ended, we reset the agent's position, and a new target position was randomly generated. This function defined the dynamics of the environment and how the agent's actions affected it.

## 2. Rewards Distribution:-

- In reinforcement learning, designing an appropriate reward system is crucial for training an agent effectively. In this project, the reward system was designed to incentivize the agent to reach the target position while avoiding the obstacle.
- To achieve this objective, we designed the reward system in a specific way. The first element of the reward system was a very high positive reward of +10000, which the agent would receive if it reached the target position successfully. This reward was designed to motivate the agent to reach the target position as quickly as possible.
- The second element of the reward system was a very high negative reward of -10000, which the agent would receive if it collided with the obstacle. This negative reward was designed to discourage the agent from colliding with the obstacle and encourage it to navigate around it.
- The third aspect of the reward system was the negative reward based on the distance between the agent position and the target position. By giving a negative reward that increases as the agent moves farther away from the target position, the agent is incentivized to come as close as possible to the target position. However, there are different ways to design this negative reward based on distance. The second approach was to use the maximum possible distance between the agent position and the target position and subtract the current distance between them from it. This would also encourage the agent to move closer to the target position.

However, it has a potential drawback that the agent may not actually reach the target position, as it can get more points by staying near the target position but not reaching it. To avoid this issue, the first approach was used in this project, where a negative reward is directly proportional to the distance between the agent position and the target position. This ensures that the agent is incentivized to come as close as possible to the target position and will have to reach the target position otherwise, it will receive a negative reward.

- Finally, we added a constant negative reward of -5 to ensure that the agent did not simply move around randomly and would try to reach the target position as soon as possible. This negative reward was designed to discourage the agent from taking unnecessary actions and motivate it to make the most efficient moves toward the target.

Overall, the reward system we designed for our reinforcement learning algorithm was essential in training the agent to reach the target position while avoiding the obstacle. The reward system provided the necessary motivation for the agent to make the right decisions based on the environment, and the negative rewards discouraged the agent from making incorrect decisions. By designing a reward system that was specific to our problem, we were able to successfully train an agent that could reach the target position while avoiding obstacles.

# Code explanation for training the agent:-

- ***For Discrete Action space and Discrete Observation Space***

  The environment is defined in a custom class called AgentEnv which is imported at the beginning of the code. First, an instance of the environment is created using ***env = AgentEnv()***. Next, a *Proximal Policy Optimization (PPO)* agent is created using the PPO class from the ***stable_baselines3*** library. The PPO agent is initialized with the ***'MlpPolicy,'*** which specifies a multi-layer perceptron neural network policy. The *verbose* parameter is set to 1, which will print out information about the training progress.

  The PPO agent is trained for different timesteps for different environments according to the requirement using the learning method. The trained agent is then

saved to a file using the save method, which specifies the path where the trained model will be saved. After training, the trained agent is loaded back into memory using the load method from the PPO class. The ***env*** parameter is set to the AgentEnv instance to ensure that the loaded agent is compatible with the environment.

Next, the agent is tested for a different number of episodes using a loop that iterates over the number of episodes. Within each episode, the environment is reset using the reset method, which returns the initial observation. The agent then takes actions using the predict method, which takes the current observation as input and returns the action to take.

The observation, reward, and done status are returned from the step method, which takes the action as input. The observation is then updated, the score is calculated based on the reward, and the render method is called to display the environment with the current score. If the episode is done, the loop breaks, and the score for the episode is printed on the console. Once all episodes are completed, the average score is calculated and printed to the console. Finally, the environment is closed using the close method.

Overall, this code uses reinforcement learning to train an agent to navigate an environment while avoiding obstacles and then tests the trained agent to evaluate its performance.

- ***For continuous Action space and continuous Observation Space***

  All the code remains the same as in the discrete case to train the agent. In this case, we used the Proximal Policy Optimization (PPO) algorithm with the 'MlpPolicy' and also included hyperparameters to address the issues we encountered during training. Gamma, the discount factor used to calculate

  expected rewards in the future, was set to 0.999. The learning rate of the PPO algorithm was set to 0.00003, while the number of times the policy network was trained on the collected data, n_epochs, was set to 10. These hyperparameters were chosen based on experimentation and may require fine-tuning for optimal performance.

  We extended our previous work by creating a custom environment with a continuous action space and continuous observation space, which was trained using the PPO algorithm. By modifying variables and implementing a step function that rewarded and penalized the agent based on its performance, we were able to train the agent to navigate the environment and reach the target.

During the training process, we encountered challenges in optimizing the agent's performance. One challenge was the difficulty in selecting appropriate hyperparameters for the PPO algorithm. We experimented with different values for gamma, learning rate, and n_epochs to find the optimal values that would result in the agent learning the optimal policy. Additionally, we noticed that the agent sometimes got stuck in a local minimum and struggled to escape from it. We mitigate this issue by using a large number of epochs to ensure that the agent has sufficient opportunities to explore the environment and learn from its mistakes. But still, there were some problems in training the agent that we have addressed in the discussion part.

## Discussion and Conclusion:-

This project focuses on training an agent to navigate through a simulated environment toward a designated target while avoiding obstacles. The environment is composed of an agent image, an obstacle image, and a target image. The observation space for the agent included six elements, which are the x and y coordinates of the position vectors for the agent, obstacle, and target. The action space represents the possible actions that the agent can take to reach the target position.

Overall, the project aims to train an agent that could navigate through an environment in a way that mimics human-like behavior. This is done by providing the agent with a set of observation and action spaces, which allow it to learn how to avoid obstacles and reach the target efficiently. The success of the project is measured based on the ability of the agent to reach the target while avoiding obstacles consistently.

In the initial phase of our project, we experimented with a discrete observation space where the agent's position, obstacles, and target location could be anywhere within the defined environment but with discrete numbers. Additionally, the agent's actions were also limited to discrete movements, specifically up, down, left, and right by one step. We chose this approach as a starting point to evaluate the effectiveness of our reinforcement learning algorithm in navigating through the environment while avoiding obstacles.

We conducted multiple experiments to evaluate the performance of the agent trained for various numbers of steps, ranging from 10k to 1M. We observed that as the number of training steps increased, the performance of the agent also improved. However, we also observed a phenomenon called overtraining, where the agent's behavior becomes unstable and erratic after a certain number of training steps. To measure the performance of the agent, we considered the total rewards obtained by the agent at the end of each task. Through experimentation, we discovered that training the agent for 4-5 lakh steps was sufficient to achieve complete training for the navigation task while avoiding obstacles. These findings suggest that while increasing the number of training steps may

improve the performance of the agent, there is a limit to the benefits of increased training, beyond which overtraining may occur, resulting in reduced performance.

Furthermore, we tested the trained model by varying the obstacle, target, and agent starting positions while keeping the other parameters constant. Specifically, we trained the model on a particular obstacle position and then used the same trained model for different obstacle positions. We repeated the same process for target and agent starting positions. In the discrete action and observation space, we observed that the agent achieved success rates of more than 80%, indicating that the trained model is robust enough to generalize to different scenarios. This result indicates that the agent has learned to navigate the environment effectively while avoiding obstacles and that the trained model is capable of making decisions in different scenarios with the same level of success. But, when both the target and obstacle was randomized together the agent was reaching the target but sometimes it takes too much time which can also be due to the hidden effects of the hyper-parameters.

After achieving considerable success with the discrete environment, we moved to a continuous environment, where the observation space was continuous, meaning that the position of the agent, obstacles, and target could be anywhere within the specified environment. Additionally, the actions available to the agent were also continuous, ranging from -1 to 1, with negative actions providing the ability to step back. We trained the agent for different numbers of steps, and while the agent performed well with less training, it started to behave erratically as the number of training steps increased. Specifically, we noticed that the agent would freeze or get stuck upon touching the lower boundary of the environment. These observations suggest that the transition to a continuous environment requires a different approach to training the agent, as the increased complexity requires a more nuanced and adaptive approach to learning.

The challenge of training the agent in a continuous environment was a significant roadblock in our project, and we spent considerable time exploring different methods to overcome this challenge. We attempted to modify the reward function to incentivize the agent to avoid the lower boundary, but the problem persisted. We also attempted to impose a high negative reward when the agent hit the boundary, but this approach did not lead to a satisfactory solution. Despite our efforts, we were unable to achieve consistent and reliable training of the agent in a continuous environment, highlighting the need for further research and development in this area.

To address the challenge of training the agent in a continuous environment, we adopted a different strategy. Previously, we were using the velocity of the agent as an action, which meant that if the agent were at position (x, y) and took an action of (x1, y1), the updated position of the agent would be (x + x1, y + y1). We manually prevented the agent from moving outside the observation space by considering any action that would result in the

agent leaving the boundary as zero. However, we realized that this was not an optimal approach and did not utilize the full capabilities of the environment. To overcome this, we utilized the properties of the Pygame environment by using the rect.right, rect.left, rect.up, and rect.down functions of Pygame to obtain the boundaries of the agent. By utilizing these properties instead of manually controlling the actions, we were able to solve the problem and achieve successful training of the agent in a continuous environment.

Additionally, we noticed that the agent sometimes got stuck in a local minimum and struggled to escape from it. We mitigate this issue by using a large number of epochs to ensure that the agent has sufficient opportunities to explore the environment and learn from its mistakes.

Although we were successful in resolving the freezing effect, we noticed that the agent's movement was still not smooth. The agent's motion appeared fuzzy, with the agent moving left and right in a jerky manner. We attributed this to the fact that the action space was based on velocity. To address this issue, we modified the action space to be based on acceleration, where the agent's acceleration would update its velocity, and the velocity would then update the agent's position. This resulted in much smoother agent movement. However, this modification did not necessarily improve the agent's ability to navigate the environment effectively, as it still struggled to reach the target position and avoid obstacles.

To improve the agent's performance in navigating the environment, we needed to tune the hyper-parameters of the PPO policy algorithm. In our constrained time for this project, we focused on understanding 2-3 hyper-parameters that would have the most significant impact on the agent's performance. Through experimentation and trial-and-error, we were able to find a set of hyper-parameters that resulted in the agent effectively navigating the environment, avoiding obstacles, and reaching the target position with high success rates.

One of the hyperparameters we considered was gamma, which determines how much the agent is allowed to explore. By default, gamma was set to 0.99, which meant that only 0.01% of the time was allocated for exploration. For episodic tasks, we recommended a gamma value close to 1, as there should be limited exploration and more exploitation to complete the task quickly. However, for continuous tasks, we suggested a gamma value of less than 0.5, as there should be more exploration compared to exploitation to achieve optimal performance. Tuning gamma can significantly improve the performance of the RL agent and is an essential part of hyperparameter optimization.

When the gamma parameter was set to 0.99, the agent's performance was not satisfactory as it seemed to wander aimlessly and did not reach the target position or avoid colliding

with the obstacle. To address this issue, we increased the gamma parameter to 0.999 since our task was episodic in nature. This led to a significant improvement in the agent's performance as it started to avoid obstacles when trained for over 100,000 steps. However, it was still unclear whether the agent was successfully reaching the target position or simply moving randomly around the environment. Further analysis and tuning of hyperparameters were needed to optimize the agent's performance.

To evaluate the performance of the agent further, we converted our task into a continuous task from an episodic one. In the previous setting, the task would end when the agent reached the target position or collided with an obstacle. However, in a continuous task, the agent is expected to perform the task indefinitely. We changed the gamma parameter and trained the agent again. The agent was able to avoid obstacles and reach the target position, but it was not staying at the target position continuously. This result indicates that there is still more work to be done in fine-tuning the hyper-parameters of the PPO policy to achieve optimal performance.

It is important to have a proper understanding of how different hyper-parameters may impact the final result and how they are used in neural networks in any of the RL policies before making changes to them. For instance, the learning rate determines how much the weights of the neural network will change after each iteration of training. If the learning rate is too high, the network may converge to a suboptimal solution or even diverge, while if it is too low, the convergence may be very slow. Similarly, epochs determine how many times the agent will learn from a batch of data during each training iteration. Increasing epochs can lead to better convergence but may also lead to overfitting while decreasing it can lead to underfitting. Therefore, it is important to have a proper understanding of these hyper-parameters and how they work in neural networks to make informed decisions about their values.

In conclusion, the development of an RL agent to navigate a maze was a challenging task that required careful consideration of hyper-parameters and the design of the environment. We encountered several issues related to the action space, the movement of the agent, and hyperparameter tuning. By making adjustments to the action space and using environment properties, we were able to solve the initial problems with the agent's movement. We also explored the impact of gamma, the learning rate, and epochs on the agent's performance. Ultimately, our experiments highlighted the importance of understanding hyper-parameters and their impact on the training process. With further optimization, it is possible to create an RL agent that can effectively navigate a maze and avoid obstacles.

## Acknowledgments:-

We would like to express our sincere gratitude to Professor Madhu Vadali and Suraj Boorate for their invaluable guidance and support throughout this project. Their expertise in the field of reinforcement learning and their willingness to provide insightful feedback has been instrumental in the successful completion of this project. Their continuous encouragement, patience, and constructive criticism have been a great source of motivation for us. We would also like to thank them for providing me with access to the necessary resources and tools to carry out this research. We are deeply grateful to have had the opportunity to work with such knowledgeable and dedicated mentors.

## References:-

- https://www.coursera.org/specializations/reinforcement-learning
- https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
- https://stable-baselines3.readthedocs.io/en/master/index.html
- https://blog.paperspace.com/creating-custom-environments-openai-gym/
- https://www.youtube.com/watch?v=PYylPRX6z4Q&ab_channel=RobertMiles