**Department of Computer Science, IIT Bombay**

**AUTUMN SEMESTER JUL-NOV 2024**

**Deterministic Safe Shield Synthesis for the playground arena in a Partially Observable Environment**

Tipu Sultan1, Hasmita Kurre1

**Name : Md Tipu Sultan, Hasmita Kurre**

**Roll No. : 23V0010, 23D0385**

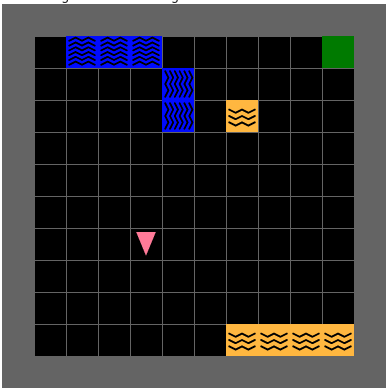
**Academic Programme : Formal Methods in Machine Learning (CS-781)**

# **Abstract:**

This paper presents a deterministic shield synthesis approach for the "Playground" game, a partially observable grid-based environment where an agent must navigate safely to a goal while avoiding hazardous "lava" tiles. The environment poses significant challenges due to presence of Lava at random locations, which introduce a chance that the agent’s intended moves may result in unintended game end situation. Additionally, intermittent partial observability complicates the agent’s task, as lava tiles may be hidden from view for a specified number of steps. Our objective is to ensure the agent reaches the goal without encountering these hazardous tiles, relying on a deterministic shield that overrides risky actions to prevent the agent from moving into unsafe states. Our methodology utilizes formal reachability analysis to evaluate each potential action’s safety, allowing the shield to either validate safe moves or reroute the agent to avoid high-risk areas. This deterministic approach guarantees robust navigation by continuously maintaining a “safe envelope” around the agent, ensuring that its movement remains clear of hazards even in the presence of environmental randomness and limited visibility. We tested our synthesis framework on various grid layouts and configurations of partial observability, demonstrating that the deterministic shield significantly reduces the frequency of hitting the lava, allowing the agent to complete the course safely across a wide range of game scenarios. These results highlight the potential of deterministic shield synthesis to support safe agent navigation in uncertain environments, providing valuable insights for applications in safety-critical control and autonomous systems.

# **1.0 Introduction**

The "Playground" game serves as a simplified yet rich testbed for exploring deterministic decision-making under uncertainty in agent-based systems. This game unfolds on a grid, where an agent must navigate from a start position to a designated goal, avoiding hazardous "lava" tiles that represent unsafe states. The agent’s task is complicated by the random nature of the environment: when the agent moves, there is a chance it will slide onto a lava adjacent to it. This randomness in movement adds a unique layer of complexity, making it difficult for the agent to ensure safe traversal across the grid. Ensuring the agent's safety despite these challenges calls for a sophisticated control strategy—namely, deterministic shield synthesis, which serves as a guardian layer, intercepting risky actions to prevent the agent from stepping onto dangerous tiles. In contrast to probabilistic shields, which rely on probability-based models to make decisions under uncertainty, deterministic shield synthesis emphasizes robustness by enforcing strict safety constraints. This approach is particularly valuable in scenarios where the agent’s behavior must be predictable and free from stochastic elements, as is often required in critical safety applications. Here, we investigate how deterministic shield synthesis can ensure that the agent reaches its goal without stepping on lava tiles, even in the presence of movement uncertainty and partial observability.



**Figure 1 Shield Synthesis for Playground arena**

# **2.0 Problem Description**

The Playground game takes place on a grid, with certain tiles marked as "safe" and others as hazardous "lava" tiles. The agent’s primary goal is to navigate across this grid while avoiding lava tiles, which would end the game upon contact. The grid is partially observable, meaning the agent has incomplete information about the tile layout and potential hazards. Specifically, in certain configurations, the agent may lack visibility of lava tiles for \*m\* out of every \*K\* steps, simulating intermittent sensor failure or communication lag. Given these constraints, our objective is to design a deterministic shield that governs the agent’s actions, intercepting moves that could place the agent in danger. The shield must ensure safe navigation by either confirming that a planned move will not land the agent on a lava tile or adjusting the action to avoid this risk. This design demands a careful balance between flexibility and safety, as the shield must dynamically adjust its recommendations based on the agent’s current location, visibility conditions, and potential slipping effects. Additionally, the shield must be adaptable to different \*m/K\* ratios, accounting for varying degrees of partial observability without compromising safety.

# **3.0 Deterministic Shield Synthesis for the "Playground"**

The concept of shield synthesis in reinforcement learning, particularly in safety-critical environments, is central to ensuring that an agent can operate safely while still learning optimal policies. In the context of the "Playground" game, where the agent must navigate a grid and avoid stepping on hazardous "lava" tiles while contending with a slippery environment that introduces randomness into movement, a deterministic shield synthesis approach offers a structured method to ensure the agent’s safety under all conditions. Building on the framework presented in the paper \*Safe Reinforcement Learning via Shielding\* by Alshiekh et al., the deterministic shield synthesis approach guarantees safety by predefining a set of safe states and actions for the agent. In contrast to probabilistic shielding, which relies on predicting future states based on uncertain or partial information, deterministic shielding works by constructing a shield that actively intervenes to enforce safety constraints whenever the agent’s actions would lead it into a hazardous state, such as stepping onto a lava tile.

The key idea behind deterministic shield synthesis is to use formal methods, particularly reachability analysis, to evaluate the safety of each possible action in the environment. The shield is designed to continuously check the agent’s current state and the intended action. If the action would lead the agent into a state that is not safe (e.g., moving onto a lava tile), the shield alters the agent's decision, ensuring that only safe actions are executed. This method guarantees that the agent is always steered away from unsafe situations, even when faced with unexpected movement caused by the environment's slippery nature.

In the context of the "Playground" game, the shield operates in two main phases: (1) Safety Assessment – where the shield evaluates each potential move to determine if it would lead to an unsafe state, and (2) Action Modification: where, if the intended move is unsafe, the shield overrides it by selecting a safe alternative. This design ensures that the agent’s learning process can continue without violating safety constraints, allowing it to explore the environment and learn effective policies without the risk of fatal errors. The deterministic shield synthesizes a conservative set of safety constraints and systematically enforces these constraints, ensuring the agent is never in an unsafe state, regardless of the stochastic elements introduced by the slippery mechanics of the game. This robust safety framework allows the agent to function in partially observable and partially controllable environments while still satisfying safety conditions, a crucial aspect of reinforcement learning in real-world applications where failure can result in catastrophic consequences.

# **4.0 Anticipated Challenges**

The main challenge in this deterministic shield synthesis problem is to maintain safety despite the agent’s inability to observe the environment occassionally. Partial observability adds an additional layer of difficulty, as the shield must synthesize a reliable decision-making strategy based on incomplete knowledge of the grid. For example, when visibility of lava tiles is temporarily obscured for ‘m’ out of ‘K’ steps, the shield must either rely on prior observations or implement cautionary measures, such as avoiding areas of the grid where the presence of lava tiles is uncertain. This requires a synthesis process that can enforce deterministic constraints without full visibility, ensuring that the agent’s path remains clear of hazards even when observation gaps limit its understanding of the environment. In conclusion, by adapting the shielding framework to the "Playground" game, we need to demonstrate how deterministic shield synthesis can be applied to environments with randomness and partial observability. The shield guarantees that the agent avoids unsafe states under all circumstances, while still allowing it to continue learning from the environment. This approach not only enhances the agent's safety but also serves as a model for deploying safe reinforcement learning techniques in more complex, real-world systems where safety is paramount.

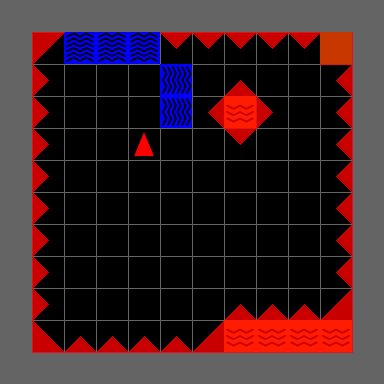
# **5.0 Code for Safe Shield Synthesis:**

Tempest and tempestpy for implementing safe RL are developed and maintained by the Trusted AI group at Graz University of Technology for shield synthesis in grid like environments. Existing Tempest\_In\_Action codes were co-developed by Steffen Pranger and team. Tempest project is developed to foster the research of shielded reinforcement learning. Alongside the synthesis tools, MinigridSafe and Minigrid2PRISM were combined to allow the modelling of uncertainty and additional, potentially adversarial, actors into grid world environments. Tempest uses a discrete, stochastic model of an environment to synthesize a shield. In order to automate the integration of shields into the training of agents in MinigridSafe environments, Minigrid2PRISM have been provided which maps the internal dynamics of any MinigridSafe into a symbolic model in the PRISM modelling language. Using these models, shielded training can be fully automated using tempestpy and Minigrid2PRISM. Tempest in action project packs all the relevant source code and uses docker to build a virtualized environment in order for you to try out the framework.

**Tempest In Action:**

Playground is a simplistic environment, showcasing the typical workflow where the aim of the agent is to reach goal i.e. green grid in a best possible way by accumulating maximum rewards without hitting the lava. A safe shield needs to be designed which ensures the agent reach its goal without hampering the safety specifications.

The main method of this script declares a safety specification that states that the agent should not visit any lava state, a shield value used for training, and a shield comparison type of absolute. After instantiating the MinigridSafe environment, shields for different values are created. Since this is a fully deterministic environment, only the shield with a safety threshold of 1.0 has been used. Once the shield has been synthesized, the symbolic model and a visual representation of the shield is printed.



**Figure 2 Deterministic Shield Synthesis for Playground arena**

A shield ensuring safety in this environment, blocks the agent from entering a lava state. A red triangle means that the agent is not allowed to move forward on the adjacent tile. Note, that since the model does not include actions that would make the agent move into a wall, the shield does not allow forward movement in these states.

The provided Python script PlayGround.iypnb and utils.py demonstrates how to train a reinforcement learning model using Stable Baselines3 (SB3) with action masking and shielding in a MiniGrid environment. The installation of the docker file and code interpretations have been discussed in Appendix-A at the end of the paper.

**5.0 Methodology**

To address these challenges, we propose a deterministic shield synthesis approach based on formal verification techniques that systematically evaluate potential action outcomes. The shield uses a model of the grid layout that integrates known information about the distribution of safe and hazardous tiles, along with the likelihood of slipping. Based on this model, it intercepts any movement action that could lead to a lava tile, either by rerouting the agent to an alternative safe action or by pausing its movement until a safer option is available. Our synthesis process builds on reachability analysis, which assesses whether a given state (such as a tile near a lava hazard) is reachable from the agent’s current position. The shield uses this analysis to limit the agent’s actions, ensuring that its trajectory remains within a “safe envelope” that minimizes the risk of slipping into hazardous areas. By maintaining this safe trajectory, the shield effectively mitigates the risks posed by the game’s inherent randomness, while deterministic constraints allow the shield to operate without probabilistic inference.

**Proximal Policy Optimization (PPO):**

PPO is a reinforcement learning (RL) algorithm developed by OpenAI, which is designed to optimize the policy of an agent in an environment through policy gradient methods. PPO is widely used because it strikes a good balance between ease of use, sample efficiency, and performance. PPO is an on-policy reinforcement learning algorithm, which means that it learns from the data generated by the current policy (as opposed to off-policy algorithms like Q-learning, which can learn from data generated by past policies). PPO is designed to improve the stability and reliability of training by making small, conservative updates to the policy at each step. The Basic Idea of PPO is to improve on the Trust Region Policy Optimization (TRPO) algorithm by simplifying its implementation and making it more efficient. The key idea behind PPO is to optimize the policy with multiple small steps rather than large updates, which ensures that the new policy does not deviate too much from the old one. In PPO, the goal is to update the policy based on the advantage of the action taken (which indicates how good that action was compared to other actions), while ensuring that the update is not too large, avoiding instability.

PPO uses a clipped surrogate objective function to enforce that the policy update stays within a "safe" region (i.e., the change in the policy does not move too far from the old policy). PPO provides a stable and robust method for policy optimization, avoiding the instability and variance issues associated with other policy gradient methods.

Proximal Policy Optimization (PPO) is an effective reinforcement learning algorithm known for its balance of stability, efficiency, and simplicity. It is widely used in various applications, including games, robotics, and control tasks. Its key strength lies in the clipping mechanism, which ensures that the updates to the policy are small enough to avoid large destabilizing changes, leading to more reliable and stable learning.

**6.0 Experiments & Results:**

**6.1 Shield Synthesis for Fully Observable MDP:**

Several experiments were conducted using the existing code to investigate and understand the process by which the safe shield for the arena [1] is computed from the safety specification, with temporal logic being applied to fully observable systems (FO-MDP). In these experiments, multiple simulations were run with the training parameter set to 1, which corresponds to a deterministic shield for the arena.

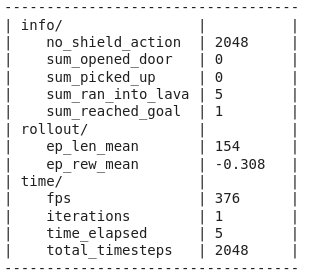


Figure 3 Shield Synthesis for Fully Observable System

It was found that, with this configuration, the shield during the training was consistently synthesized safely, effectively preventing the agent from colliding with the ‘Lava’ (i.e., `run\_into\_the\_lava = 0`). Furthermore, the performance of the shield during the training process was observed by making minor adjustments to the code, and the results were documented in the figures below. These modifications provided additional insights into how the shield evolves and performs in various scenarios.

|  |
| --- |
|  |
| **Figure 4 Events vs Timesteps for fully observable systems** |
|  |
| Figure 5 Loss function for fully observable systems during training |

From the figures above, it is clearly demonstrated that the shield successfully synthesizes the system while adhering to all safety specifications, ensuring that no collisions with the 'Lava' occur at any point during the training phase. Also, with each training the episode mean and episode length to reach the desired value keeps on decreasing i.e. decreases from 400 to nearly 25 which indicates it is learning faster and more efficient way. This consistent behaviour highlights the effectiveness of the shield in maintaining safety throughout the process. The shield's ability to prevent any undesirable outcomes, such as running into the lava, further reinforces its reliability in achieving safe system operation. This outcome underscores the robustness of the shield synthesis, confirming that the safety requirements are met without fail throughout the training process.

**6.2 Shield Synthesis for Partially Observable MDP:**

Next, a series of additional experiments were conducted to investigate the performance of the shield when it is unable to visualize certain events due to external factors such as sensor failures, black-outs, or other disruptive conditions. These experiments aimed to simulate situations where the shield’s visibility is limited, leading to potential challenges in synthesizing a safe shield. The visibility frequency was varied between 1 and 0.1. A visibility frequency of 1 corresponds to a fully observable system where the shield can detect all environmental actions, while a frequency of 0.1 means the shield can only observe 10% of the environment’s actions. These observed actions are randomly distributed, meaning the shield may miss consecutive instances or see actions at unpredictable intervals. This introduces a significant level of complexity to the shield synthesis process, as the shield must still ensure safety despite the lack of full observability.

During the unobservable phases, the shield initially selected random actions, as it did not have sufficient information to enforce specific safety actions. As a result, the shield had no control over the agent’s actions, and only a subset of safety actions were available. **Minor modifications in the code both in mail file and utils.py has been made to incorporate the same.**

The results of these experiments, showing how the shield performed under varying visibility conditions, are presented in the figures below:

|  |
| --- |
|  |
| Run into Lava vs Different Visibility frequencies |
|  |
| Sum reached goal vs Different Visibility frequencies |
|  |
| Rewards length & Rewards mean vs Different Visibility frequencies |

Interestingly, the system was able to safely synthesize the shield up to a visibility frequency of 0.8. To ensure the reliability of these findings, the code was run multiple times for each visibility frequency, and the results were consistently in line with this conclusion. It was observed that for the current arena and system configuration, safe shield synthesis could be reliably achieved up to a visibility frequency of 0.8.

In the next set of observations, we focused on comparing the shield’s performance when trained with visibility frequencies greater than 0.8. We plotted the training parameters for visibility frequencies of 0.8, 0.9, and 1 to understand how the shield training changes with increasing visibility. The results of this training phase are shown in the figure below:

|  |
| --- |
|  |
| Run into Lava vs Different Visibility frequencies (Training of Agent) |
|  |
| Sum reached goal vs Different Visibility frequencies (Training of Agent) |
|  |
| Rewards length & Rewards mean vs Different Visibility frequencies (Training of Agent) |

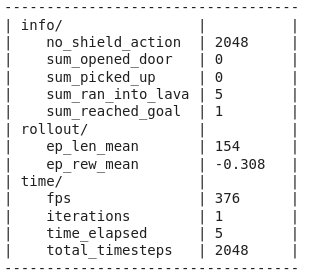
In conclusion, the experiments conducted demonstrate that the shield synthesis process is effective in maintaining safety specifications under varying levels of visibility. When the system is fully observable (visibility frequency of 1), the shield consistently prevents collisions with the 'Lava' and successfully adheres to all safety requirements. As the visibility frequency is reduced, the complexity of shield synthesis increases due to the shield’s limited ability to observe environmental actions. Despite these challenges, the system was able to maintain safe shield synthesis up to a visibility frequency of 0.8, proving that the shield can still operate effectively in partially observable conditions.

However, once the visibility frequency drops below 0.8, the shield's ability to synthesize safely becomes less reliable, particularly when visibility is as low as 0.1. This highlights the importance of maintaining a certain level of system observability to ensure consistent and safe behavior. The experiments also showed that the shield’s training performance improves as the visibility frequency increases, with higher visibility frequencies (0.8, 0.9, and 1) leading to better shield performance and more accurate safety enforcement.

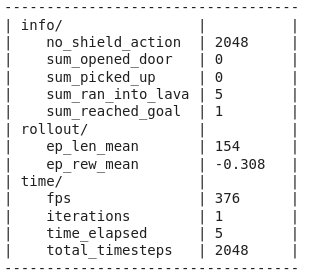
**6.3 Safe Shield Synthesis for Partially Observable MDP:**

The next approach that was explored for safe shield synthesis in partially observable systems involved introducing a mechanism where the agents are forced to remain in their current state whenever the shield is unable to observe the environment. This strategy was designed to ensure that, during periods of unobservable conditions, the agent would not take any potentially unsafe actions. The implementation of this approach was achieved by adding an extra action in the `def get\_allowed\_actions\_mask(actions):` function. This additional action represents the "do nothing" choice, which is selected whenever the shield is unable to view the environment. When this happens, the shield invokes the last action in the list, effectively instructing the agent to remain in its current state.

By incorporating this "stay" action, the shield ensures that the agent does not inadvertently perform any unsafe actions during the periods of reduced observability. This approach introduces a form of conservatism in the shield’s decision-making, minimizing the risk of violations of the safety specifications in partially observable situations. The safe shield synthesis process was successfully carried out with this modification, and the results of this approach are shown in the figures and tables below, demonstrating the effectiveness of this strategy in ensuring system safety despite the limited visibility.



**Shield training**



**shield training**

**Table-1: Statistics of Shield synthesis for POMDP**

|  |  |  |
| --- | --- | --- |
| Visibility Freq. | No. Of Safety Violations | Rewards of Gaols |
| 0.1 |  |  |
| 0.2 |  |  |
| 0.3 |  |  |
| 0.4 |  |  |
| 0.5 |  |  |
| 0.6 |  |  |
| 0.7 |  |  |
| 0.8 | 4 | 4 |
| 0.9 |  |  |
| 1 |  |  |

From the experiments that were conducted for safe shield synthesis of the Partially-Observable Systems by modifying the existing code, it can be clearly seen that we could synthesis the shield upto the visibility frequency of 0.5 where shield is able to guide the agent safely to reach the goal albeit slowly and with lesser accumulated rewards.

**7.0 Conclusion**

The Playground game provides a valuable context for exploring deterministic shield synthesis in a partially observable and probabilistically influenced environment. Our approach ensures the agent’s safe navigation across a grid laden with hazards, despite the inherent risk of slipping and the constraints of intermittent observability. By emphasizing deterministic decision-making, this shield synthesis framework demonstrates a robust solution for environments requiring predictable and safe agent behavior, which is essential in real-world applications where randomness and partial observability frequently intersect with strict safety requirements.

This study contributes to deterministic shield synthesis research by outlining methods to adaptively respond to partial observability and probabilistic influences, offering insights for broader applications in safety-critical navigation and control systems.

**KEY LINKS**

• Project demo video

• Project code

**ACKNOWLEDGEMENTS**

We, the authors, extend our heartfelt appreciation to Mr. Rohit Kapur for his extraordinary effort to solve the issue to help us with the docker installtion successfully. We thank Prof. Dr. Supratik Chakraborty along with our colleagues/classmates from entire CS-781 (Formal Methods in Machine Learning) class who provided useful insights and expertise that greatly assisted the project work although they may not agree with all of the interpretations/conclusions of this paper. This paper reflects our collective dedication to advancing knowledge in our field, and we feel privileged to have worked alongside these talented individuals.

# REFERENCES

1. “Safe reinforcement learning via shielding” by M. Alshiekh, R. Bloem, R. Ehlers, B. K ¨onighofer, S. Niekum, and U. Topcu; In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018.

2. “TEMPEST - Synthesis Tool for Reactive Systems and Shields in Probabilistic Environments” by Stefan Pranger1, Bettina K ¨onighofer, Lukas Posch, and Roderick Bloem; May,2021; DOI: 10.48550/arXiv.2105.12588

**Appendix-A:**

**Tempest In Action:**

Installation Procedure:

**Option-1:**

This option explores installing the whole respiratory locally on machine without installing it on PC.

Step-I: Install docker on your machine (UNIX environment)

Step-II:

1. git clone

<https://git.pranger.xyz/sp/Tempest_in_Action>

2. cd Tempest\_in\_Action

3. sudo docker build -t tempest\_in\_action

This will build the docker image tempest\_in\_action. The image contains all the necessary binaries to automate the workflow for shielded RL training in MinigridSafe environments, all with a convenient jupyter notebook frontend.

Step-III:

Start the container using the docker\_run\_jupyter.sh script within the docker

Several notebooks are there to play with out of which we selected playground arena.

However this option-1 gave us an error in the step-13 as follow:

**Option-2:**

Step-I: Download install\_script.zip folder

Step-II: install docker in local machine

Step-III: run docker\_run\_ .sh command

Step-IV: Once downloaded open cmake list file of tempest\_in\_action and run the code insdie the contained one by one.

Step-V: use this command to launch Jupyter Notebook on server:

**Key Components of the code:**

We have taken “Playground” arena as our problem statement where agent has to navigate to the green coloured grid without hitting lava. Shiels synthesis has two parts in this one is Playground.iypnb file and another is utils.py.

The code has followings:

**1. Imports:**

- Various libraries for gym environments, SB3, MiniGrid, and utility functions.

- Custom utility functions for shield handling and logging.

**2. Masking Functions**:

- `mask\_fn` creates an action mask based on the environment's state.

- `nomask\_fn` is a fallback function returning no mask (all actions allowed).

**3. Main Function: `main`:**

Environment Setup: The environment `MiniGrid-Playground-v0` is created and wrapped with additional wrappers like `RGBImgObsWrapper` and `ImgObsWrapper` for image observation handling.

Shield Configuration: A shield configuration is set up with a formula (safety requirement). Based on this, a `MiniGridShieldHandler` is created, which integrates shielding during the training process.

Action Masking: Depending on the configuration (`Training` or `Disabled`), the environment is wrapped with an action masker that either applies a custom mask (`mask\_fn`) or does not mask actions (`nomask\_fn`).

Model Training: A `MaskablePPO` model (from SB3) is set up and trained on the environment for a specified number of steps.

**4. Shielding:**

If the shielding configuration requires it (`ShieldingConfig.Training`), a shield handler is created with various shield values (e.g., 0.9, 0.99, 0.999, etc.), and the shield is applied to the environment during training.

A shield overlay is generated for visualization.

**5. Callbacks:**

The training process uses the `InfoCallback` for logging additional information during training.

**6. Logger:**

The logger is set up to output logs to the console using the `HumanOutputFormat`.

**7. Key Custom Classes and Functions:**

MiniGridShieldHandler:

This class handles the logic for managing the shield during training and evaluation.

MiniGridSbShieldingWrapper:

This wrapper integrates the shield handler with the environment, ensuring that the agent's actions are constrained by the shield.

create\_shield\_overlay\_image:

This function generates and displays an image showing the shield overlay on the environment.

ShieldingConfig:

Enum or configuration that defines the shielding setup (Training, Disabled, etc.).

**What’s Happening in the Code:**

Environment Setup:

The environment is wrapped to provide image observations and action masking functionality.

Shielding Integration

A shield is created based on the given safety formula. The environment is then updated with this shield using a custom wrapper.

Model Training:

A PPO model is trained using the masked environment, with the `MaskablePPO` algorithm. The model learns to avoid dangerous actions based on the shield configuration.

Visualization:

The shield overlay image is created to visualize the safe zones in the environment, helping ensure that the agent operates within safe areas.