Journal on Artificial Intelligence

Tech Science Press

DOI: 10.32604/jai.2022.039703
Article

Implementation of Strangely Behaving Intelligent Agents to Determine Human Intervention During Reinforcement Learning

Christopher C. Rosser, Wilbur L. Walters, Abdulghani M. Abdulghani, Mokhles M. Abdulghani and Khalid H. Abed*

Department of Electrical & Computer Engineering and Computer Science, Jackson State University (JSU), Jackson, 39217, MS, USA

*Corresponding Author: Khalid H. Abed. Email: khalid.h.abed@jsums.edu

Received: 12 February 2023; Accepted: 21 March 2023; Published: 24 May 2023

Abstract: Intrinsic motivation helps autonomous exploring agents traverse a larger portion of their environments. However, simulations of different learning environments in previous research show that after millions of timesteps of successful training, an intrinsically motivated agent may learn to act in ways unintended by the designer. This potential for unintended actions of autonomous exploring agents poses threats to the environment and humans if operated in the real world. We investigated this topic by using *Unity's Machine Learning Agent Toolkit* (*ML-Agents*) implementation of the Proximal Policy Optimization (PPO) algorithm with the Intrinsic Curiosity Module (ICM) to train autonomous exploring agents in three learning environments. We demonstrate that ICM, although designed to assist agent navigation in environments with sparse reward generation, increasing gradually as a tool for purposely training misbehaving agent in significantly less than 1 million timesteps. We present the following achievements: 1) experiments designed to cause agents to act undesirably, 2) a metric for gauging how well an agent achieves its goal without collisions, and 3) validation of PPO best practices. Then, we used optimized methods to improve the agent's performance and reduce collisions within the same environments. These achievements help further our understanding of the significance of monitoring training statistics during reinforcement learning for determining how humans can intervene to improve agent safety and performance.

Keywords: Artificial intelligence; AI safety; reinforcement learning; human-in-the-loop; intrinsic motivation; unity; simulations; human-machine teaming

1 Introduction

Recent mistakes and accidents caused by Artificial Intelligence (AI) result in a range of consequences for human users, from implications in crime [1] to, more gravely, immediate physical threats [2]. Solutions for creating AI that minimizes the risks leading to mistakes and accidents generally involve Reinforcement Learning (RL), reward engineering, and careful design and selection of utility functions. However, there is still a need for solutions that leverage fast or real-time human insight or intervention. This is considered challenging for reasons including, but not limited to, the costs of testing applications in the highly unpredictable real world and the unreliability or unavailability of real humans during AI tasks.

Simulating complex real-world scenarios in platforms such as *OpenAI*, *Arcade Learning Environment*, and *Unity* is an effective method for addressing some of the RL challenges as described in [3–5]. Determining when or how a real human can efficiently intervene during an intelligent agent's training process remains an active research area. Most RL platforms allow the user to visually observe the RL process, making it possible to witness an agent learn strange or undesired behaviors in real-time. These behaviors are overlooked, dismissed, or characterized in other way—"critically forgetful" in [6] and "surprising" in [7] are two examples. To the best of our knowledge, these misbehaviors have not qualified in terms of training statistic correlations or goal-to-collision ratios.

In this research, we train extremely curious intelligent agent under conditions that cause them to act undesirably and observe trends and drastic differences in the training statistics in cases where the agents experience higher-than-average collisions. We present the results of the following achievements: 1) eight experiments in which agents are trained in *Unity* learning environments with various curiosity strengths, rewards, and penalties, 2) a goal-collision metric for measuring the performance and how well the trained agent is accomplishing the required tasks, and 3) Proximal Policy Optimization (PPO) + Intrinsic Curiosity Module (ICM) training statistics correlations. Our experiments address the following three research questions: 1) *Can agents learn undesired or dangerous behaviors in less than 1 million timesteps*? 2) *Is the goal-collision metric a good environmental indicator of intelligent agent misbehavior*? 3) *Do the strongly correlated training statistics reveal information not mentioned in PPO+ICM best practices*?

2 Related Work

Literature in the following research areas describes the prospective risky intelligent agent actions in hypothetical environments.

2.1 AI Safety

Researchers and scientists have long cautioned against the emergence of superintelligence, prompting some to set forth guidelines by which AI must not harm humans as in [8]. The circumstances and hypothetical scenarios by which AI can behave in ways unintended by human researchers is categorized and discussed in great detail in [9]. We adopt these characterizations of unintended AI behavior and refer to them throughout this text as misbehavior or mischief. Many proposed experiments to increase safety and reduce risk are translated into *Gridworld* learning environments [10], where potential agent misbehavior is also discussed. Other potentially unsafe AI behaviors include an agent's willingness and unwillingness to cooperate with humans in [11,12]. Our previous work used an adaptive neuro-fuzzy algorithm to generate the required real-time control signals and avoid obstacles for a two-wheel drive system [13].

2.2 Intrinsic Curiosity Module (ICM) for Exploration

ICM addresses sparse environmental rewards by allowing intelligent agents to reward themselves for exploring new states as formalized in [14]. ICM is extended in [15] for increasing the efficiency of exploring novel states and is implemented in robotic tasks environments and in [16] to establish an attention-oriented flavor of curious exploration. ICM is clarified in [17] to create a reward bonus to overcome "couch-potato" task executed by intelligent agents.

2.3 Human-in-the-Loop Reinforcement Learning

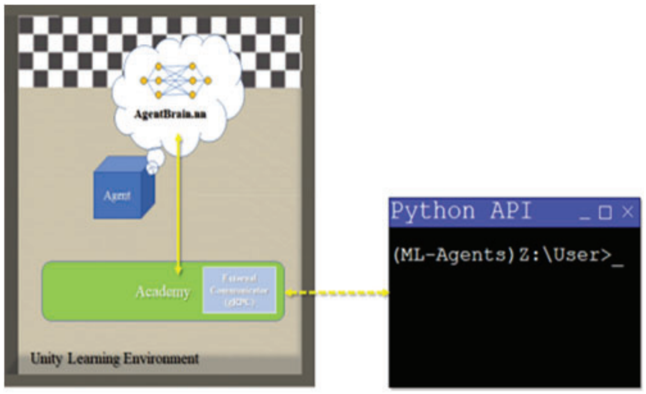
Human-in-the-loop RL methods incorporate human supervision or other forms of input to assist the agent in learning to avoid actions that may result in critical mistakes. Some implementations utilize only a limited amount of real human input, which is then used to train an agent to learn the human's input preferences before completely taking over as the human-in-the-loop, such as in [18,19]. Other human-in-the-loop RL methods include environment monitors and action blockers, as described in [20,21], and hard-coded rules based on human logic such as in [22,23].

3 Methodology

Application Programmable Interface (API) has been used along with the algorithm to train and observe the actions driven by neural networks in *Unity* learning environments.

3.1 Unity + Machine Learning Agents Toolkit (ML-Agents)

Unity allows users to quickly build and develop various 2D and 3D games and Augmented Reality/Virtual Reality experiences in a graphical user interface called the *Unity* Editor. We use *Unity* Editor of *Unity* 2018.3.6f1, to adjust the number of instances of three built-in learning environments and modify and implement custom metrics in the agent and academy C# script files. We build the learning environments into executables and launch the training sessions externally through the Python API available in *ML-Agents* 0.8.2. An open-source high-performance Google Remote Procedure Call (*gRPC*) has been used as a framework. The *gRPC* framework serializes learning environment information using remote procedure calls, in which the serialized information is passed as vector inputs to the neural networks in the *ML-Agents* implementations of baseline RL algorithms as pictured in Fig. 1. More info about *gRPC* can be found in [24].



```
(ML-Agents)Z:\User>_
```

Figure 1: *ML-Agents* learning environment

3.2 Learning Environments

The experiments have been conducted in three of ML-Agent's benchmark learning environments that represent autonomously exploring agents of varying complexity. The environments are all walled-in platforms on which agents must explore to reach their goals. The agents, represented by blue cubes, are equipped with Raycasts, which outputs a value associated with the agent's proximity to the object that was detected by rays. Raycasts are reminiscent of Lidar methods for measuring distances using laser light and are explained in greater detail in [25]. When the agent accomplishes the goal, or the expected number of timesteps passes, the training episode will end immediately whether the goal was achieved or not. Then, the agents, blocks, and goals will be respawning in unplanned places in the environment to start a new episode.

For the *PushBlock* environment, there is one agent, and detectable objects are the wall, goal, and block. The behavior parameters have been set with 14 Raycasts returning an observation vector of 70 variables. The agent has six actions, which are Turn clockwise, turn counterclockwise and turn in four different face directions. The agent aims to push the orange block to the checkered pattern. The rewards and penalties are +1.0 for moving to golden brick, and −0.0025 for each step, respectively. The benchmark mean reward has been set to 4.5.

For the *Pyramid* environment, there is one agent, and detectable objects are wall, goal, block, and stone. The switching status has been set to be either off or on. The agent behavior parameters were 21 Raycasts returning an observation vector of 148 variables. Four actions (turn clockwise, turn counterclockwise, move forward, and move backward) have been used. The agent's goal is to press a switch to spawn a pyramid, find and knock over the pyramid, and move to a targeted brick that falls from the top. The rewards and penalties are +2.0 for moving to golden brick, and −0.001 for each step, respectively. The benchmark mean reward has been set to 1.75.

For the *Hallway* environment, there is one agent, and detectable objects are the orange goal, red goal, orange block, red block, and wall. The behavior parameters have been set with 5 Raycasts returning an observation vector of 30 variables, and the agent has four actions which are turn clockwise, turn counterclockwise, move forward, and move backward. Agent's goal is to move to the checkered area next to the color that is the same as the block's color. The rewards and penalties are +1.0 for moving to golden brick, and −0.001 for each step, respectively. The benchmark mean reward has been set to 0.7.

In each learning environment, as in Fig. 2, we implement a goal counter (G^), a collision counter (C), and a ratio between the two (1) to be observed during Play Mode. The ratio qualifies how successfully an agent reaches its goals without crashing with any related objects in the environment and is defined as:

$$GC_{Ratio} = \frac{G^\wedge}{(G^\wedge + C + \alpha)} \tag{1}$$

where (α) value was set to 0.001 in (1). The value of α prevents errors in calculations resulted from division-by-zero and ensures that the ratio is working in a test runs and producing the best path for the agent to move without hitting objects. We keep the agent agnostic to these three metrics by not adding them to the agents' vector observations.

3.3 Experiments

Eight test cases have been done, as illustrated in Table 1. The following six conditions have been examined: 1) Low/Zero Curiosity Strength, Small Rewards, and Large Penalty. 2) Low/Zero Curiosity Strength, Large Rewards, Small Penalty. 3) Max Recommended Curiosity Strength, Small Rewards, Large Penalty. 4) Max Recommended Curiosity Strength, Large Rewards, Small Penalty. 5) Very High Curiosity Strength, Small Rewards, Large Penalty. 6) Very High Curiosity Strength, Large Rewards, Small Penalty.

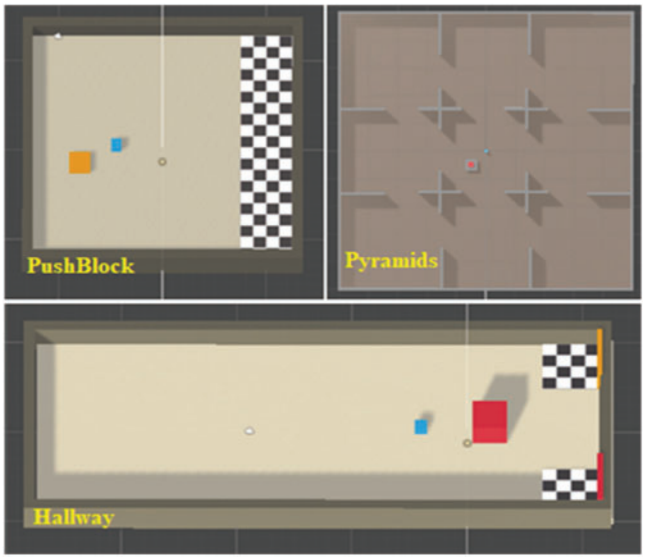


Figure 2: Top-down views of learning environments in *Unity* editor

Table 1: Curiosity strengths and rewards for test cases 1–8

	PushBlock		Hallway			Pyramids		
Case	Curiosity strength	Rewards	Curiosity strength	Rewards	Penalties	Curiosity strength	Rewards	Penalties
1	Disabled	0.5	Disabled	1	−100, −0.1	0.01	2	−100
2	Disabled	50	Disabled	100	−1, −0.1	0.01	200	−1
3	0.1	0.5	0.1	1	−100, −0.1	0.1	2	−100
4	0.1	50	0.1	100	−1, −0.1	0.1	200	−1
5	1.0	0.5	1.0	100	−1, −0.1	1.0	2	−100
6	1.0	50	1.0	1	−100, −0.1	1.0	200	−1
7	10.0	0.5	10.0	1	−100, −0.1	10.0	2	−100
8	10.0	50	10.0	100	−1, −0.1	10.0	200	−1

3.4 Proximal Policy Optimization + Intrinsic Curiosity Module (PPO+ICM)

The PPO has been used to train our agents based on its performance against other well-known policy optimization methods, as demonstrated in [26]. In *ML-Agents* implementation of PPO, we enable the use of ICM by simply setting a Boolean value in the trainer configuration file and adjusting the curiosity strength and other hyperparameter values as necessary. The PPO+ICM used in this work was developed using *Tensorflow/Keras* Back-End Neural Networks (BENN). The main rule of the ICM is indicating the next status and the predicted next status. According to the amount of error between the next states and the predicted next status, the reward signals that encourage agents to explore new states will be generated. More details about generating the reward signal are presented in [15] and pictured in Fig. 3.

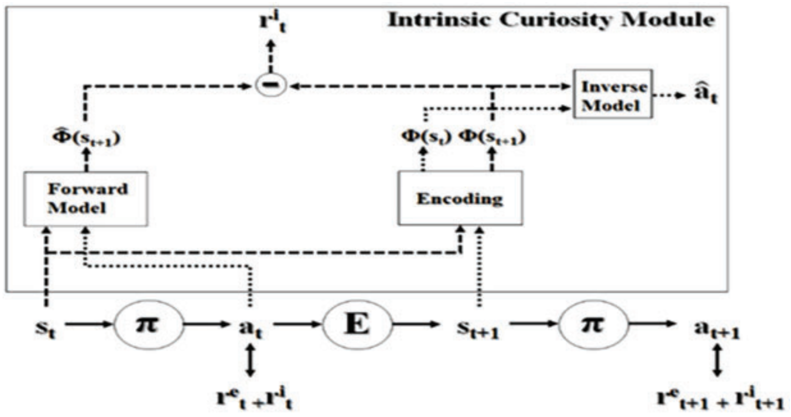


Figure 3: ICM computes the error of next states and predicted next states

We use PPO+ICM hyperparameter values within the recommended ranges per the best practices in [25]. We deviate from the best practices in these experiments by using curiosity strength values in and outside the recommended range for several experiments. Table 2 lists the parameter values used in all experiments.

Table 2: The training parameters of the PPO + ICM

Parameter	Value
γ	0.990
Beta (β)	0.01
α	0.0003
Time-horizon (τ)	64–128
Max-step (S_Max)	150,000
λ	0.950
Curiosity-strength	0.01–10

Where (γ), (β), and (λ) are *Gamma, Betta*, and *Lambda*, which are the learning hyperparameters for the PPO+ICM. Alpha (α) represents the learning rate. The number of hidden units was set to be (128–512), the buffer size to be (1024–2048), and the batch size to be (128). Eight operations have been implemented for each learning environment based on the rewards and hyperparameters in Tables 1 and 2, respectively, with each build containing fifteen instances of its learning environment. We then launch the training externally from an *Anaconda Prompt* with the following command:

```
mlagents-learn ../trainer_configuration_file
--env=LearningEnvironment --run-id=UserCreatedID –train
```

During training the step, mean cumulative reward, standard deviation, and elapsed time prints in the *Anaconda* Prompt every 1000 steps for the *PushBlock* cases and every 2000 steps for the *Hallway* and *Pyramid*s cases. The PPO+ICM training statistics are written to comma separated value files at the same frequencies and can be monitored in a browser, as demonstrated in [27]. In a separate window, also during training, we visually observe the agent performing actions in the learning environments. Training lasts for 150,000 steps.

At the end of each case, a trained neural network model is saved in the same directory as the corresponding executable. We name the neural networks after the learning environment and case number in which they were trained i.e., "PushBlock1.nn" for *PushBlock* case 1. We return to each learning environment in the *Unity* Editor and attach the resulting trained neural network to the agents. For each case, we then observe agent behavior during Play Mode for 60, 180, and 300 s, respectively. We use average goals, collisions, and ratios to restrict our focus to the cases in which the agents produce the best number of collisions throughout the Play Mode. We record .mp4 video footage of all cases during Play Mode with *Unity* Recorder.

Finally, we plotted the strongest positively and negatively correlated training statistics and focus our analyses on cases where trained agents cause the most collisions: *PushBlock* cases 2 and 3, *Hallway* cases 4, 6, and 7, and *Pyramid*s cases 7 and 8. We compared the performance of agents attached with *ML-Agents* pre-trained neural networks in Table 3 to the performance of agents in cases 1 through 8 to identify the cases chosen for a deeper analysis, listed in Table 4.

Table 3: Goals, collisions, GC-Ratio averaged over 30 runs

Learning environment	Goals	Collisions	GC-Ratio
PushBlock	**78.6**	**4.1**	**0.9452**
Hallway	52	2.57	0.9567
Pyramids	10.4	14.6	0.4074

Table 4: The improved experiments case

Cases	Penalties	End function	Reward conditions
Case 1	0	0	0
Case 2	1	0	0
Case 3	1	1	0
Case 4	1	1	If GC-Ratio > 0.7
Case 5	1	1	If PC < -0.7

4 Extended the Work

4.1 Performance Optimizations Methods for Agents

We develop the following methods for determining highly curious and reckless agents:

Collision Penalty: We gave the agent penalty (−1) when it collided with non-goal.

Switch-Off: We call the ML-Agent function *EndEpisode()* immediately after the agent collided with non-goal objects. This function will terminate the running episode if the agent touches non-goal objects.

Goal-to-Collision Ratio (GC-Ratio) Reward Condition: The rewards will be conditioned according to GC-Ratio. If the GC-Ratio is less than 0.7, the agent will receive a big reward. If not, the agent will receive a small reward.

Pearson Correlation (PC) Rewards Condition: The rewards will be conditioned according to the current PC value. If this PC value is less than −0.7, the agent will receive a big reward. If not, the agent will receive a small reward. The person correlation value has been computed every 11 episodes.

4.2 Extended the Dynamics of the Environments

Three new environments have been added for training and testing as shown in Fig. 4.

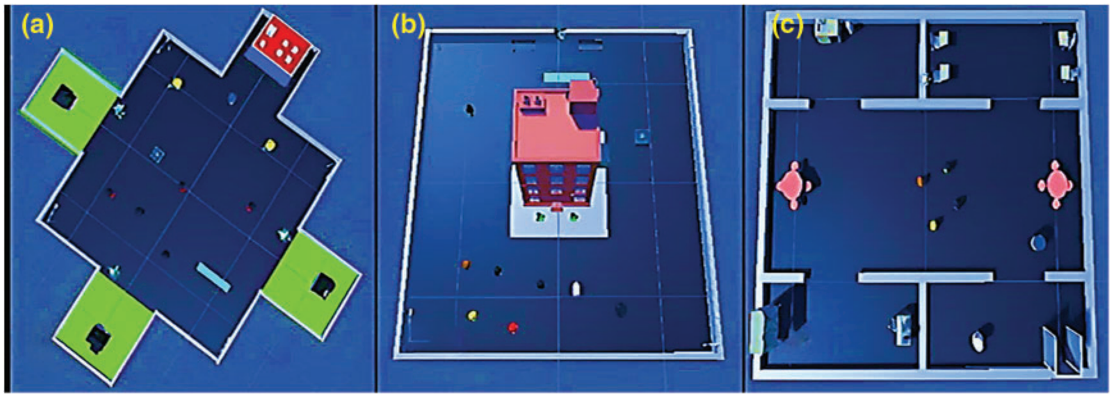


Figure 4: The three environments

We add to the environment: Navigation Meshes (*NavMesh*) [25] is a *Unity* built-in tool that uses A* algorithm [28]. This algorithm helps us generate different walking paths to the goal for the agent in every episode. We used *NavMesh* to create walking paths for all agents. This will add a dynamic feature in the running environment, and the agent will take a different route to the goal every episode. To challenge the agent in the environments, we added a group of objects (Road Blocks, Target, and Goal) in the environments. *NavMesh* has been used to add multiple groups of agents in the environments.

5 The Extension of the Agents

5.1 Agent Script

We extend the agent's script by adding: 1) Collision penalties, 2) Add more Raycasts, 3) Add goal counter, 4) Add collision counter, 5) Add GC-Ratio counter, 6) Add Episode counter. All these additional features have been added to every agent's script in each of the three experiments.

5.2 Agents' Behavior Parameters

We added to the agent: 18 Raycasts return an observation vector of 486 variables over a 180-degree "field of vision". Detectable Objects: Object1, object2, npcAI, floor, wall, target, and goal. The agent has seven actions: Turn clockwise, turn counterclockwise, move in four different face directions, change the agent color to red, and set the agent speed to zero. The agent aims to push the decorated block to the cyan rectangular object. Rewards/Penalties: +10.0 if the block touches the rectangular object and $-(1/textrmMaxStep)$ each step.

The updated agents have been trained and evaluated in the three environments. To track the agents' performance with goals, collisions, and GC-Ratio values. We will present the performance of these agents in the results section.

6 The Extended Experiment

To verify the performance of the proposal methodologies, we tested the agents in the three environments. Then, we evaluate the methodologies with the same agents in two training environments to examine the performance improvements for the agents in environments.

6.1 Environments Experiment

The Benchmark Environment Experiment shown in Table 4 is to train the agents in the *PushBlock*, *Hallway*, and *Pyramid* environments using the PPO+ICM algorithm.

Table 4 provides information for our improved methods used in the three experiments. The cases column points out the first five cases in Table 1. We focus our improvements on the first five cases (case 1 to case 5) from Table 1, which lists eight cases. The value 0 in this table represents False, and 1 represents True. The Penalties column represents whether there is punishment in this case or not. The End Function column corresponds to whether, in this case, the *EndEpisode* function is used or not. The Reward Condition column corresponds to the condition for giving the reward. In case 4, the GC-Ratio must be greater than 0.7 to reward the agent. If not, the agent will receive a small reward. In case 5, the agent must have Person Correlation (PC) value less than -0.7. If not, the agent will receive a small reward.

7 Results

The results have been presented in two parts. Part-1 to show agents' performance in the *PushBlock*, *Hallway*, and *Pyramid* environments related to Table 3. Part-2 shows the improved methods that could reduce collisions and improve agents' performance in all three environments. We have results for these five cases.

7.1 Part 1

RQ1: Can agents learn undesired or dangerous behaviors in less than 1 million timesteps?

During the training process, the agents learned undesired behaviors within 150,000 steps. Either a high reward or a high penalty was given to the agent. In the Play Mode, the agent execution has been evaluated and can be distinguished to be "aloofness", "timidity", or "hesitation" in the *Hallway* environment and "recklessness" or "dangerous" in the *Pyramid* environment. We capture these behaviors in screenshots, as seen in Fig. 5. These are forms of misbehavior that affect the agent's capability to reach the goal and explore the environments but capture the agents' interest in exploring and reaching the goal despite large penalties.

The agent in the *PushBlock* environment seemed only slightly or not at all affected by intrinsic curiosity, likely due to the simplicity of its tasks and environment. The agent's infrequent collisions with the wall, captured in Fig. 6, occurred only during its attempts to push its target towards the goal, regardless of which trained neural network was attached.

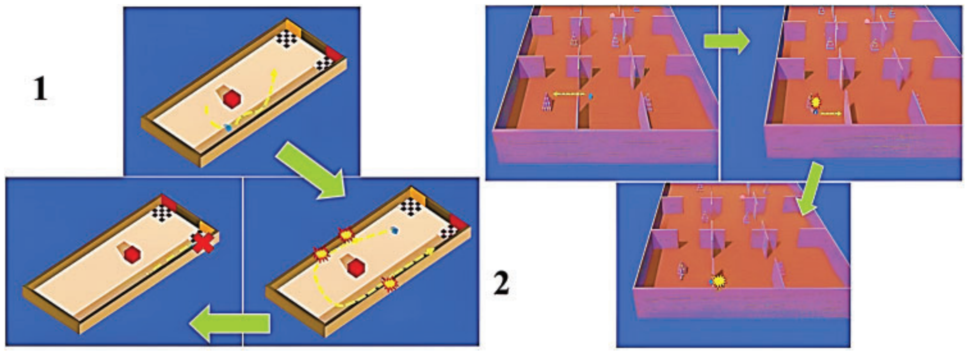


Figure 5: The observed agent action in 1) *Hallway*, 2) *Pyramid* environments

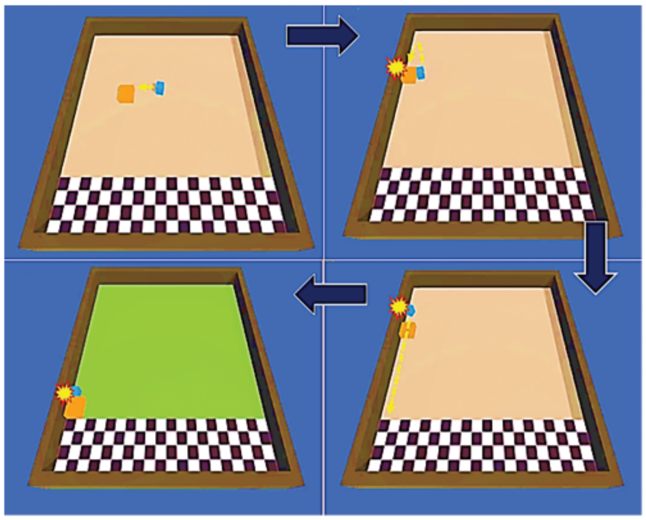


Figure 6: Agent collides with and scrapes against the wall as it completes its task

RQ2: Is the goal-collision metric a good environmental indicator of intelligent agent misbehavior?

The goal-to-collision metric efficiently reduces dangerous behaviors, as illustrated in Table 5. This is helpful because it can act as a mean for persuading a human teammate to intervene during the training process. The procedure for the metric must be revised to include extra information if it is to be used to meet the required criteria of intelligent agent actions and behaviors.

RQ3: Do the strongly correlated training statistics reveal information not mentioned in PPO+ICM best practices?

It is clear in the plotted training that the statistics were mostly synchronized with the PPO best practices. However, only some interpretations: A noticeable spike in value loss at the beginning of *Hallway* case 6 training, a growing reward that fluctuates with high frequency in *Hallway* cases number 4 and 6 and *Pyramid* case number 8, and a fluctuating entropy in *Hallway* cases 4, 6, and 7. Figs. 7 and 8 illustrate the indicated cases in the *Hallway* and *Pyramid* environments.

Table 5: Goals, collisions, and ratios for high-collision cases

PushBlock				Hallway				Pyramids			
Case	Goals	Collisions	GC-Ratio	Case	Goal	Collision	GC-Ratio	Case	Goals	Collision	GC-Ratio
2	14.3	8.67	0.69	4	0	166	0.001	7	0	372	0.001
3	45	34.33	0.71	6	0	439	0.001	8	0.334	283	0.002
—	—	—	—	7	0	611	0.001	—	—	—	—

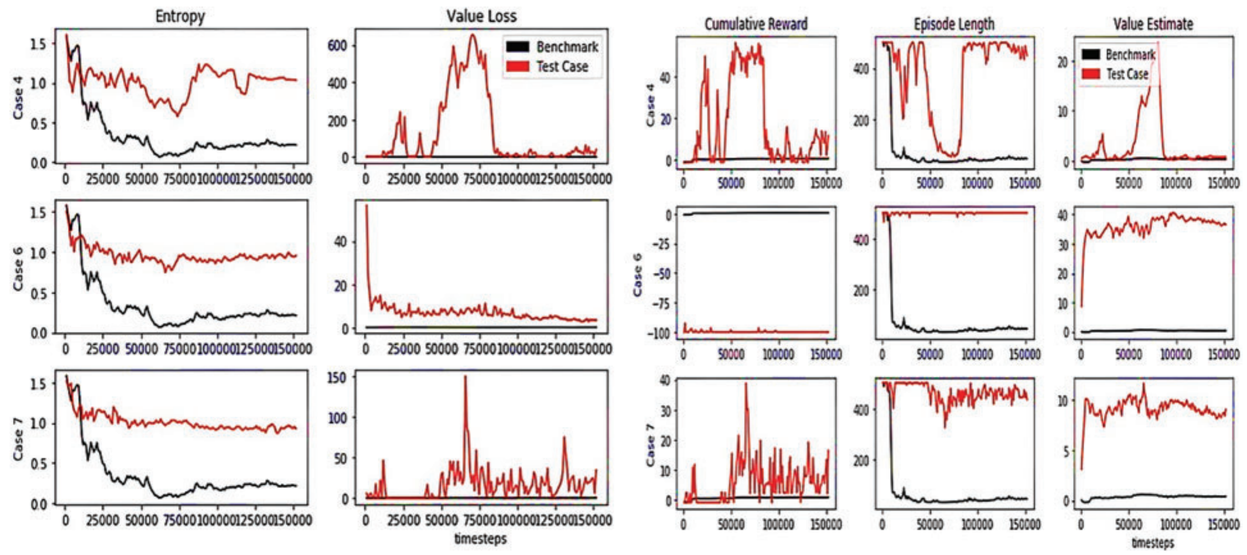


Figure 7: Highly correlated training statistics for *Hallway* cases 4, 6, and 7

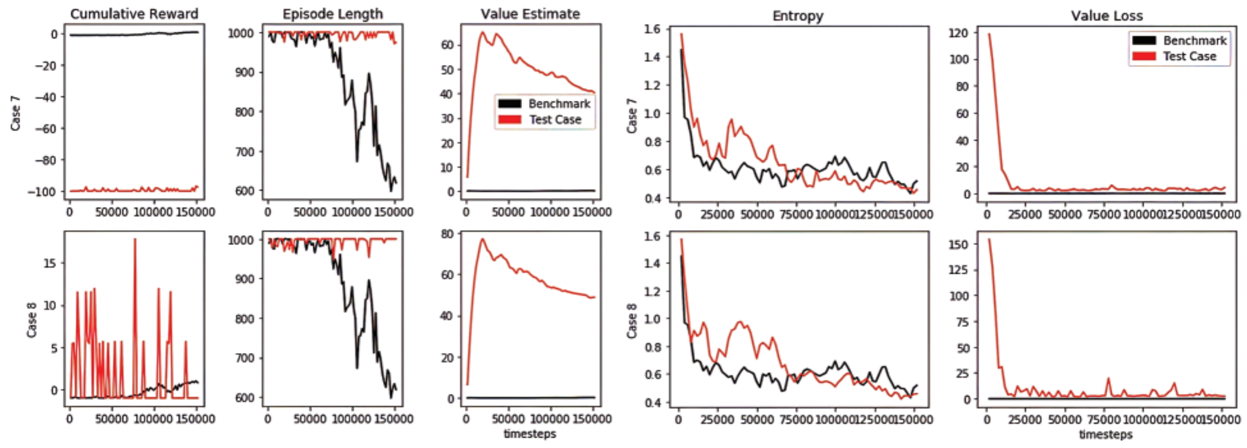


Figure 8: Highly correlated training statistics for *Pyramid* cases 7 and 8

The episode length increases as the cumulative reward decreases, validating the concept of mastered tasks requiring less time to complete. Loss value typically increases as the cumulative reward increases but is initially high and sharply declines in; for example, case 6 as the cumulative reward remained largely negative. Some fluctuations in the training indicators are anticipated providing the randomness of RL. Though, fluctuations in growing rewards are expected due to the large curiosity strength rates and more likely to be weakened by rewards production and beta hyperparameter tuning.

7.2 *Part 2*

In this part, we will clarify how the proposed methodologies improved the agents' performance for the first five cases (listed in Table 1) in the *PushBlock*, *Hallway*, and *Pyramid* environments. case 1 to case 5 were represented as method 0 to method 4, respectively. We tracked the changes in values (goals, collisions, and GC-Ratio) and recorded them in tables; then, we compared each case with Table 3.

7.2.1 Benchmark Environment Results

Push Block Environment

As shown in Fig. 9, with the first 2 million training steps using method 1, method 2, and method 3, the agent can reach the highest stable mean value of the rewards. While using method 0, method 1, method 2, and method 3 could record a stable value for the episode lengths with 2 million steps. For method 4, episode length continues to trend downwards over the remaining 8 million steps. Although method 2 is the most efficient, the agent recorded the second-lowest average number of goals and the highest average of collisions. For methods 3 and 4, all agents experienced reduced collisions (almost reached zero collisions). Table 6 shows the number of goals, collisions, and GC-Ration for all methods. The collision values in method 2, method 3, and method 4 are lower than the collision value in Table 3, which belonged to the result Part-1.

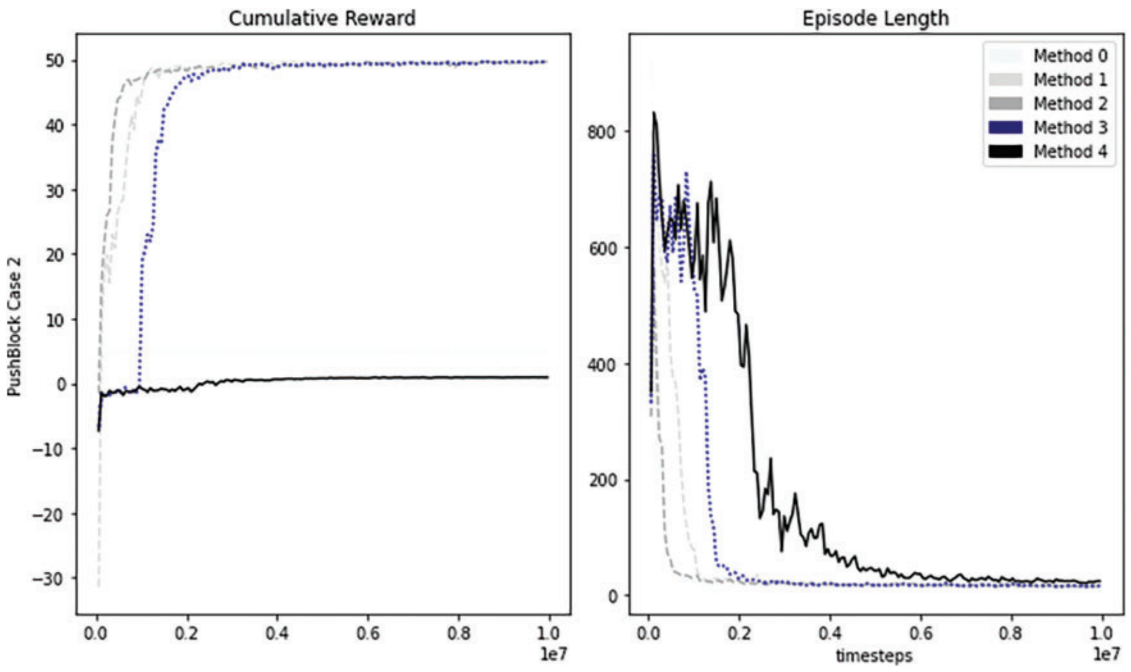


Figure 9: Performance of the agent trained with our methodologies

Table 6: Push block case 2 results averaged over 5 runs

Method	Goals	Collisions	GC-ratio
Method 0	118	7	0.9446883
Method 1	98.4	3.2	0.97306578
Method 2	70.6	12	0.83427862
Method 3	106	0.4	0.99746804
Method 4	61.2	0.4	0.99098026

Hallway Environment

This environment is more complex and difficult for the agent than the *PushBlock* Environment. Even though this is a difficult environment, the trained agent with method 2 has reached the highest steady mean value more efficiently as presented in Fig. 10. Table 7 shows that all the achieved goals by the agents in this environment are close to each other. methods 1, method 2, method 3, and method 4 experienced a reduction in collisions, and the last two methods, almost have a zero value for the collisions. We have fewer collision values in Table 7 than in Table 3, method 2, method 3, and method 4.

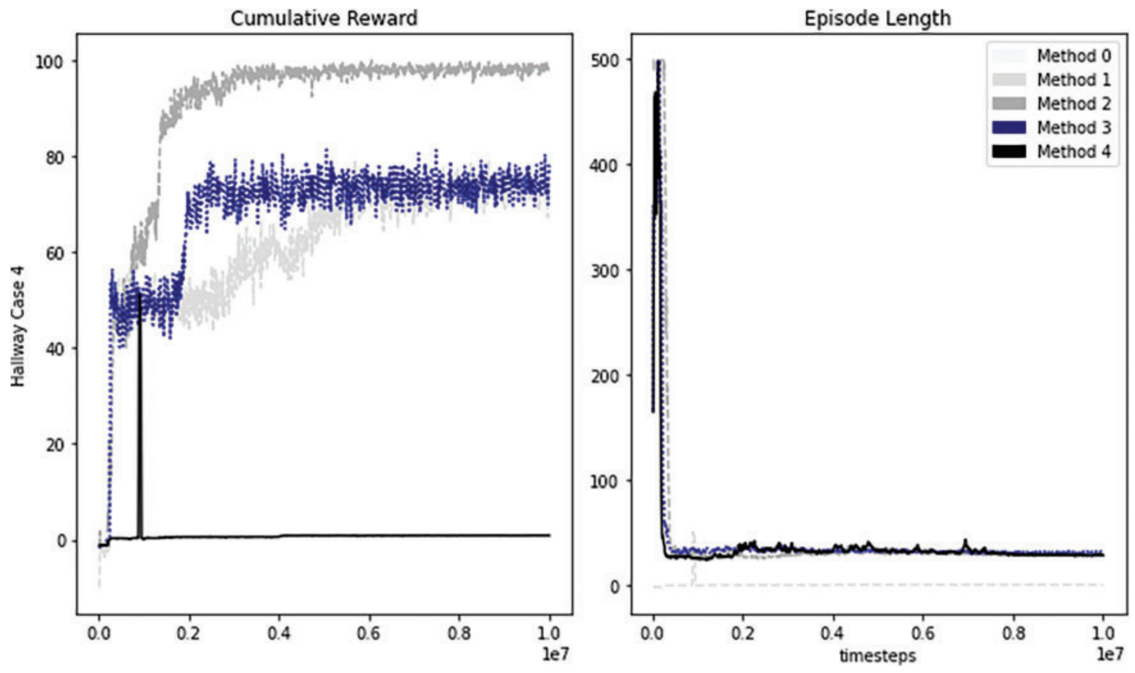


Figure 10: Performance of *Hallway* agent trained with our improved method

Table 7: Hallway case 4 results averaged over 5 runs

Method	Goals	Collisions	GC-ratio
Method 0	46.2	6	0.88957066
Method 1	38.2	4.4	0.89796346
Method 2	44.6	1.6	0.96711034
Method 3	37.8	0.4	0.9911945
Method 4	48.4	0.4	0.99305206

Pyramids Environment

This environment is the largest one, and obtaining rewards is a difficult task for the agent in this environment. As shown in Fig. 11, we can see that the reward and episode length are not stable for all methods. In Table 8, we can see that the collision's value almost doubles the goal's value, and GC-Ration's poor value of 0.4 is the highest value. if we compare the collisions result of method 1, method 2, method 3, and method 4, this method is better, because it has fewer collisions.

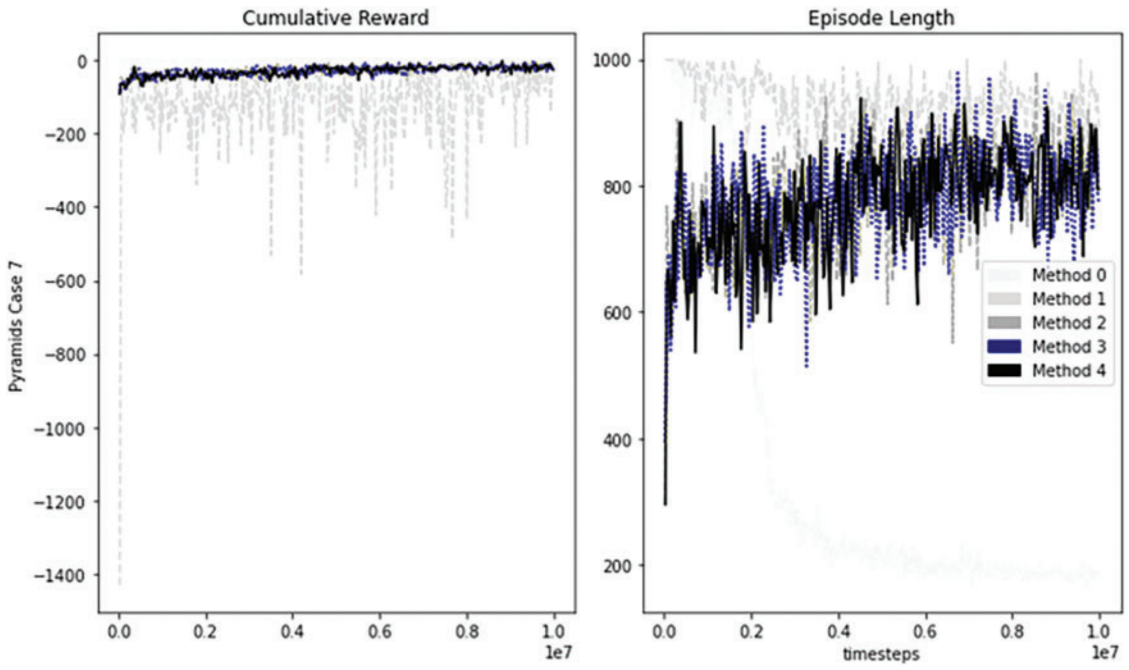


Figure 11: Performance of *Pyramid* agent trained with our methodologies

Table 8: *Pyramid* case 7 results Averaged over 5 runs

Method	Goals	Collisions	GC-ratio
Method 0	8.6	17.6	0.33330058
Method 1	0.4	10	0.2008002
Method 2	0.4	0	0.4006004
Method 3	0.4	0.6	0.2008002
Method 4	0.4	0.4	0.4006004

8 Conclusion

The benchmark learning environment results validated our assumptions that conditioning rewards on values determined by analysis of training statistics and human insight, namely the goal-to-collision, effectively reduced the collisions of highly curious agents. In contrast to the fluctuating cumulative rewards and episode lengths in the *Pyramid* environment, the smoother and quicker convergence of cumulative rewards and episode lengths in the *PushBlock* and *Hallway* environments is likely due to the smaller size of each environment. Similarly, a reduction in the average goals achieved by the agents trained with the performance optimization methods is likely due to learned behaviors that could be characterized as "caution," such that an agent more carefully navigates its environment and may take longer than the allotted playback time to complete its goals. To describe the agent's performance during evaluation over short periods. The GC-Ratio, however, remains a good metric for providing conditional rewards during agent training and describing how well an agent is reaching its goal without collisions during post-training evaluation. Although this is a clever strategy for the agent to have learned during training, it translates into dangerous behavior in novel environments in which the agent cannot end an episode forcefully. As a measurement of improved safety and comparing with the collisions in results (Part-1), the performance optimization methods show, in some cases, better trends in reducing collisions. In this work, a curiously behaving intelligent agent was trained within six specified conditions including eight cases for different periods of time. Then, we presented optimized techniques to enhance the agent's performance and decrease the collision cases in three different environments. We used a custom metric to evaluate how well the agents behave to reach the desired goals without crashing with objects. We validated the PPO's best performance and concluded that the intelligent agent learns unwanted behavior when trained with enormous intrinsic motivation value. Future work will be done using the Yolo7 algorithm for object detection, and we will investigate the possibility of employing the segmentation method to secure safe navigation.

Funding Statement: This work was partly supported by the United States Air Force Office of Scientific Research (AFOSR) contract FA9550-22-1-0268 awarded to KHA, https://www.afrl.af.mil/AFOSR/. The contract is entitled: "Investigating Improving Safety of Autonomous Exploring Intelligent Agents with Human-in-the-Loop Reinforcement Learning," and in part by Jackson State University.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] L. Dodds, "Chinese businesswoman accused of jaywalking after AI camera spots her face on an advert," The Telegraph, November 25, 2018. [Online]. Available: https://www.telegraph.co.uk/. [Accessed July 14, 2020].

[2] R. Schmelzer, "What happens when self-driving cars kill people?," Forbes, September, 26, 2019. [Online]. Available: https://www.forbes.com. [Accessed July 14, 2020].

[3] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman *et al.,* "OpenAI gym," arXiv: 1606.01540, 2016.

[4] M. Bellemare, Y. Naddaf, J. Veness and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2012. https://doi.org/10.1613/jair.3912

[5] A. Juliani, V. Berges, E. Teng, A. Cohen, J. Harper *et al.,* "Unity: A general platform for intelligent agents," arXiv: 1809.02627, 2018.

[6] W. Saunders, G. Sastry, A. Stuhlmueller and O. Evans, "Trial without error: Towards safe reinforcement learning via human intervention," arXiv: 1707.05173, 2017.

[7] J. Lehman, J. Clune, D. Misevic, C. Adami, L. Altenberg *et al.,* "The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities," arXiv: 1803.03453, 2018.

[8] D. Weld and O. Etzioni, "The first law of robotics," In: M. Barley, H. Mouratidis, A. Unruh, D. Spears, P. Scerri *et al.* (Eds.), *Safety and Security in Multiagent Systems. Lecture Notes in Computer Science*, Vol. 4324. Berlin, Heidelberg: Springer, 2009. https://doi.org/10.1007/978-3-642-04879-1_7

[9] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman *et al.,* "Concrete problems in AI safety," arXiv:1606.06565, 2016.

[10] J. Leike, M. Martic, V. Krakovna, P. Ortega, T. Everitt *et al.,* "AI safety gridworlds," arXiv: 1711.09883, 2017.

[11] N. Soares, B. Fallenstein, E. Yudkowsky and S. Armstrong, "Corrigibility," in *Proc. Workshops 29th AAAI-15 Conf. AI*, Austin Texas, pp. 74–82, 2015.

[12] L. Orseau and S. Armstrong, "Safely interruptible agents," in *Proc. 32nd Conf. Uncert. in AI*, New Jersey, USA, pp. 557–566, 2016.

[13] M. M. Abdulghani, K. M. Al-Aubidy, M. M. Ali and Q. J. Hamarsheh, "Wheelchair neuro-fuzzy control and tracking system based on voice recognition," *Sensors*, vol. 20, no. 10, pp. 2872, 2020. https://doi.org/10.3390/s20102872

[14] D. Pathak, P. Agrawal, A. A. Efros and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proc. 34th Int. Conf., ML*, Sydney, Australia, pp. 2778–2787, 2017.

[15] B. Li, T. Lu, J. Li, N. Lu, Y. Cai *et al.,* "Curiosity-driven exploration for off-policy reinforcement learning methods*," in *Proc. IEEE Int. Conf. Rbtcs. Bmtcs.*, Dali, China, pp. 1109–1114, 2019. https://doi.org/10.1109/ROBIO49542.2019.8961529

[16] P. Reizinger and M. Szemenyei, "Attention-based curiosity-driven exploration in deep reinforcement learning," in *Proc. IEEE Int. Conf. Acstcs. Spch. Sgnl. Prcsg.*, Barcelona, Spain, pp. 3542–3546, 2020. https://doi.org/10.1109/ICASSP40776.2020.9054546

[17] N. Savinov, A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys *et al.,* "Episodic curiosity through reachability," arXiv: 1810.02274, 2018.

[18] B. Prakash, M. Khatwani, N. Waytowich and T. Mohsenin, "Improving safety in reinforcement learning using model-based architectures and human intervention*," arXiv:1903.09328, 2019.

[19] S. Reddy, A. Dragan, S. Levine, S. Legg and J. Leike, "Learning human objectives by evaluating hypothetical behavior," arXiv:1912.05652, 2019.

[20] P. Mallozzi, E. Castellano, P. Pelliccione, G. Schneider and K. Tei, "A runtime monitoring framework to enforce invariants on reinforcement learning agents exploring complex environments," in *Proc. 2nd IEEE/ACM Int. Wrkshp. Rbtcs. Sftw. Eng.*, Montreal, QC, Canada, pp. 5–12, 2019. https://doi.org/10.1109/RoSE.2019.00011

[21] M. Koivisto, P. Crockett and A. Spangberg, "AI safe exploration: Reinforced learning with a blocker in unsafe environments," B.Sc. Thesis, Dept. Cmp. Sci. Eng., Univ. Gothenburg, Chalmers Univ. Tech., Gothenburg, Sweden, 2018.

[22] D. Abel, J. Salvatier, A. Stuhlmuller and O. Evans, "Agent-agnostic human-in-the-loop reinforcement learning," in *Proc. 30th Conf. Nrl. Inf. Prcs. Sys.*, 2017. [Online]. Available: https://stuhlmueller.org/papers/human-rl-nips2016.pdf

[23] E. Yeh, M. Gervasio, D. Sanchez, M. Crossley and K. Myers, "Bridging the gap: Converting human advice into imagined examples," *Advcs Cogtve. Sys.*, vol. 7, pp. 117–136, 2018.

[24] Open-source high-performance Google Remote Procedure Call (gRPC) framework. The Linux Foundation, NY, USA, 2015. [Online]. Available: https://grpc.io

[25] A. Juliani, V. Berges, E. Teng, A. Cohen, J. Harper *et al.,* "Unity: A general platform for intelligent agents," *ArXiv*, 2018. https://doi.org/10.48550/arXiv.1809.02627

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal policy optimization algorithms," arXiv: 1707.06347, 2017.

[27] C. Rosser, E. Alareqi, A. Wright and K. Abed, "Acceleration of python artificial neural network in a high performance computing cluster environment," in *Proc. 2018 Int. Conf. Data Science*, Turin, Italy, pp. 268–272, 2018.

[28] P. Mehta, H. Shah, S. Shukla and S. Verma, "A review on algorithms for pathfinding in computer games," in *IEEE Sponsored 2nd Int. Conf. on Innovations in Information Embedded and Communication Systems ICIIECS'15*, Coimbatore, India, 2015.