Journal on Artificial Intelligence

Tech Science Press

DOI: 10.32604/jai.2023.039786

ARTICLE

AI Safety Approach for Minimizing Collisions in Autonomous Navigation

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

Department of Electrical & Computer Engineering and Computer Science, Jackson State University, Jackson, 39217, USA

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

Received: 16 February 2023 Accepted: 11 April 2023 Published: 10 August 2023

ABSTRACT

Autonomous agents can explore the environment around them when equipped with advanced hardware and software systems that help intelligent agents minimize collisions. These systems are developed under the term Artificial Intelligence (AI) safety. AI safety is essential to provide reliable service to consumers in various fields such as military, education, healthcare, and automotive. This paper presents the design of an AI safety algorithm for safe autonomous navigation using Reinforcement Learning (RL). Machine Learning Agents Toolkit (*ML-Agents*) was used to train the agent with a proximal policy optimizer algorithm with an intrinsic curiosity module (PPO + ICM). This training aims to improve AI safety and minimize or prevent any mistakes that can cause dangerous collisions by the intelligent agent. Four experiments have been executed to validate the results of our research. The designed algorithm was tested in a virtual environment with four different models. A comparison was presented in four cases to identify the best-performing model for improving AI safety. The designed algorithm enabled the intelligent agent to perform the required task safely using RL. A goal collision ratio of 64% was achieved, and the collision incidents were minimized from 134 to 52 in the virtual environment within 30 min.

KEYWORDS

Artificial intelligence; AI safety; autonomous robots; unmanned systems; *Unity* simulations; reinforcement learning; RL; machine learning; *ML-Agents*; human-machine teaming

1 Introduction

Artificial Intelligence (AI) improves our daily lives by performing tasks without human intervention. Although using AI will save plenty of time and effort, the mistakes caused by AI may lead to disastrous outcomes. An autonomous agent can be programmed to search for a target or deliver a target to a certain location. This agent should be equipped with the proper hardware and software that provide it with the capabilities for safely exploring the environment and avoiding harmful accidents to the consumer. One of the most reliable solutions for such problems is equipping these autonomous systems with AI safety [1]. Using the Reinforcement Learning (RL) algorithm showed a practical performance in AI safety [2]. RL algorithm can train an agent in a specific environment and encourage the agent by giving it a reward to explore additional new areas [3]. During the training, the agent will try different actions and behaviors, such as hitting a wall or crashing a car. In a real environment, this agent can be self-driving cars, cleaning robots, military or delivery drones, and many other applications that serve the consumer. RL algorithms can enhance the agent's performance and reduce these risks by applying punishment to the agent in the training phase to train the agent to avoid any wrong or dangerous behavior using a specific punishment function. Moreover, the RL algorithm can also reward that agent for a good performance using a specific reward function [4]. The reward and punishment functions are essential to encourage the intelligent agent to learn more and warn the agent in case of unwanted behavior. To avoid physical damage in the training or testing process, Virtual Reality (VR) environments can be used [5–7] as flexible applications to train a model.

Researchers from Google, OpenAI, UC Berkley, and Stanford collaborated to identify five concrete problems in AI safety [8]: 1) avoiding negative side effects, 2) avoiding rewards hacking, 3) scalable oversight, 4) safe exploration, and 5) robustness to distributional shift. Researchers assert that these problems are attributed to three things: i) having the wrong objective function, ii) having an objective function that is too expensive to evaluate frequently, and iii) undesirable behavior during the learning process. Many of their proposed solutions to these AI safety problems involve RL. In [9], these five concrete problems were discussed and investigated Machine Learning (ML), RL, safe exploration, and AI safety.

In this paper, we used Machine Learning Agents Toolkit (*ML-Agents*) [10] to train the agent with a proximal policy optimizer algorithm with an intrinsic curiosity module [11]. Two of the five concrete problems were considered, avoiding negative side effects and safe exploration to ignore non-goal objects and evade the obstacles. *ML-Agents* were used to train the agent with Proximal Policy Optimizer (PPO) algorithm with the Intrinsic Curiosity Module (ICM) enabled (PPO + ICM). VR environment was used to train the agent to avoid obstacles or find another route around them and train it to ignore non-goal objects and evade the obstacles. This training aims to improve AI safety and minimize or prevent the mistakes that can cause dangerous collisions by the intelligent agent. To validate the results of our research, we performed four experiments. We used the *Unity* environment and *ML-Agents* in these experiments to test and execute the designed algorithm. This research will provide insight into answering the following two questions:

Q1: Are goals and collisions good metrics for choosing a model of an autonomous explorer agent?

Q2: Are 256 hidden units the best to minimize the number of collisions for an autonomous explorer agent to avoid the negative side effects?

The rest of the paper is organized as follows. In Section 2, we discuss the prior research methodology of relevant work. In Section 3, we present the proposed methodology. Section 4 presents the results and discussion, and Section 5 presents the conclusions.

2 Related Works

RL is the third class of machine learning that works to collect data by itself without human interference. This behavior helps reduce the time for a machine learning task. It works to collect data from agent behavior or action in a specific environment. This will help the model become more robust. RL shows impressive success in controlling many different games in the game of go [12], StarCraft2 [13], and Atari games [14,15]. Artificial intelligence has improved how humans live and has greatly improved several technologies. However, AI should be safe and riskless to help us more [16,17].

In our previous work, we used an adaptive neuro-fuzzy algorithm to generate the required real-time control signals and avoid obstacles for a two-wheel drive (2WD) system [18]. AI must provide stability and continuously track the agent's behavior in an environment [19]. Markov Decision Processes (MDPs), were used to reduce the risk in autonomous navigation in [20]. Another important work of the RL algorithm is Q-Learning [21], where an agent was trained to navigate safely in a kitchen environment [22].

There are many published research experiments focused on improving AI safety to lower the risk for safe exploring and navigated agents [23–25]. PPO is a state-of-art RL algorithm invented by *OpenAI* company. This algorithm can be equipped with the *ML-Agents* to design AI safety systems. *ML-Agents* help train the agent with the PPO algorithm to overcome many RL scenarios, such as hide-and-seek [26]. An agent trained with PPO can easily beat other agents in the 8-bit video game domain [27]. Another task for the PPO algorithm has been considered in a real-time environment to run an autonomous system [28].

ICM is an important model in the RL algorithm, and it is built in the *ML-Agents*. ICM is responsible for training the agents to move and explore the environment by encouraging them with rewards every time the agents reach new areas of the environment. ICM works like a rewarding system to encourage the agent to explore more [29,30]. This gives the agent an extra reward for exploring new status [11,20,23,24,31]. ICM is also considered in the design of an explorer agent to search and explore an environment [25]. We can use many applications as an environment to train our agents, such as *Unity*, *OpenAI Gym*, and *The Arcade Learning Environment*. *Unity* is considered the most responsive and flexible environment because it is user-friendly and widely used in academia, industry, and the Department of Defense. There are many extension applications available for facilitating extra functionality to *Unity*. Many researchers have used *Unity* as a training environment for their agents [32,33]. Connecting the *ML-Agents* with *Unity* gives these tools access to the *Unity* environment and perform actions and behaviors to train an agent and to build an AI model that can safely explore its environment.

3 Methodology

This section will discuss the tools, algorithms, configurations, and applications used to train our agents. The method we propose and explain in this section serves the consumer by reducing collisions, evading static and moving obstacles or objects, and minimizing the damage for the autonomous exploring agent.

3.1 Unity Environment and ML-Agents

Unity is a responsive and flexible Windows, Linux, and Mac application. *Unity* version 2019.2.15f1, *ML-Agents* release_12, *ML-Agents* Python interface 0.23.0, Python 3.7, Pytorch 1.7.1 [34], and CUDA Toolkit 11.7.1 [35], were used in this work. Fig. 1 presents *Unity* with our VR training environment. We added four people with red shirts, two farmers with yellow helmets, a police officer with a blue hat, and a target box, which is the multi-color cubic box in the red square. A white square surrounds our agent with a green shirt. We specified a goal: the rectangular region with cyan color in the orange box. We used the C# script to adjust the custom values for our environment and agent. The hardware and software used in this work include windows 11, CPU Core i7 11800H with 32 GB RAM, and RTX 3080 laptop GPU.

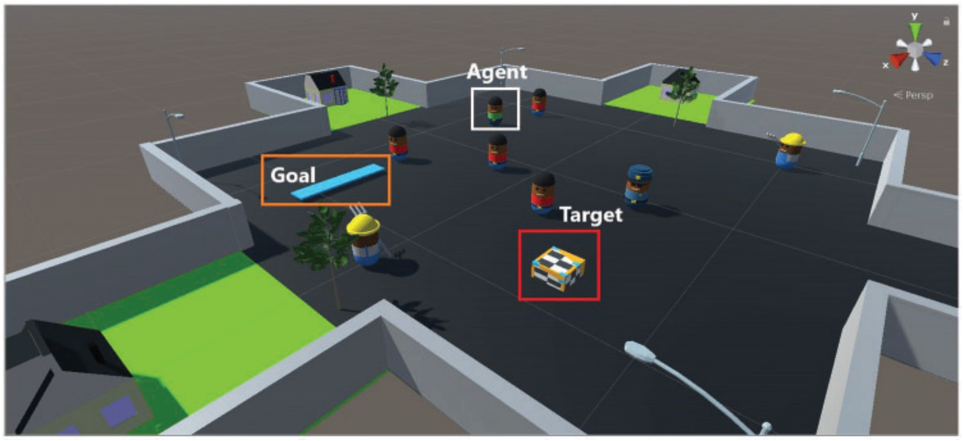


Figure 1: The VR environment

3.2 Goal Collision Ratio

Goal (G) Collision (C) Ratio (GC-Ratio) is the measurement of the goals without colliding with the objects in every round. Our agent is supposed to evade the obstacles, but he sometimes makes wrong footsteps during traveling to the target or the goal. These wrong footsteps increase the time the agent takes to complete the round. Eventually, the agent will reach the goal even though he made one or more accidents in that particular round. In this case, we specified the maximum number of movement footsteps for the agent. If the agent reaches the maximum number without arriving to the goal, the environment will be reinitialized. Then, the agent, the goal, and the target will all be initialized with a random position for each of them based on (1).

$$GC_{Ratio} = \frac{G}{(G + C + K)} \tag{1}$$

Eq. (1) is used to calculate the GC-Ratio for every round, where K is equal to 0.001 to avoid the error caused by zero division operation; C points out the number of mistakes done by the agent or collisions counter; and G is the goal counter, which represents the number that the agent successfully reached the goal. The value of GC-Ratio is unknown for the agent. It will be obtained when we run the environment and start collecting the data for the training after reaching the required data (2,048 actions or behavior). The GC-Ratio will be calculated according to the goals and collisions, whenever the agent gains a new value of goal or collision. Then, the GC-Ratio will be refreshed.

3.3 Training Environment

In the *Unity* training environment, we examined our agent initially to find the target and deliver it to the goal point, as shown in Fig. 2. In every round, the agent continued moving with the target towards the goal after any collision, and the agent eventually reached the goal with the target. Fig. 2(1) shows an example of the initial state of the environment we used in our four experiments. Firstly, the agent must search and find the target, as shown in Fig. 2(2). Then, the agent must drive the target to the specified goal point, as shown in Fig. 2(3). The other characters have a walking route that may intersect with the agent's path while the agent is trying to explore the environment to reach the goal. Fig. 2(4) shows the agent carrying the target and approaching the goal. Fig. 2(5) shows that the agent finally reached the goal with the target. This environment helps us build a robust model that works in complex scenarios to avoid obstacles. In this scenario, obstacles may block the agent's way to the target, the agent must decide to either choose a different route or wait until the path is clear. In this scenario, the path will not be simply a straight line from the starting point to the destination.

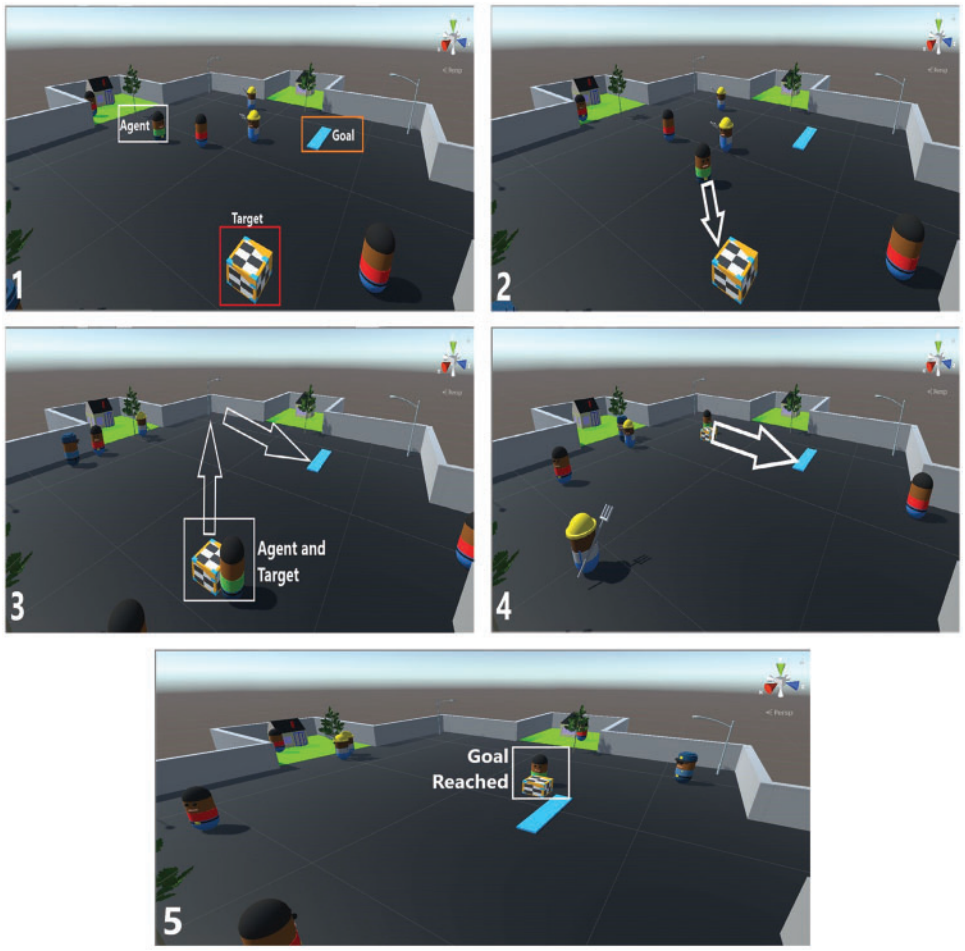


Figure 2: (1) The initial state of the environment. (2) The agent searches for the target. (3) The agent drives the target. (4) The agent drives the target to the goal. (5) The agent reached the goal with the target

This agent has two Raycasts [10], which work like a sensor to measure the distance between the agent and the nearest block. Each sensor covers 180 degrees, so we need two sensors to provide full 360-degree coverage around the agent. The feedback data of Raycast will be sent to the ICM inside the *MLAgents*, and the *ML-Agents* will make decisions based on the received data. Each iteration will be finished when the agent drives the target to the goal. If the agent finished the maximum number of movement footsteps without reaching the goal, the environment will be reinitialized. When the agent hits any blocks like walls or a character, the system will penalize the agent by giving it a collision point.

The agent's behavior in discovering the environment is illustrated in Fig. 3. This way, the ICM will know it made a mistake and try to avoid it in future rounds. With this training, we will have an intelligent agent that can detect the target and drive it to the goal by avoiding the obstacles. In Exp_1, we start the training for our model with 2 million max_steps (iterations) and 128 hidden units. The hyperparameters for PPO + ICM that we used in this training are included in Table 1.

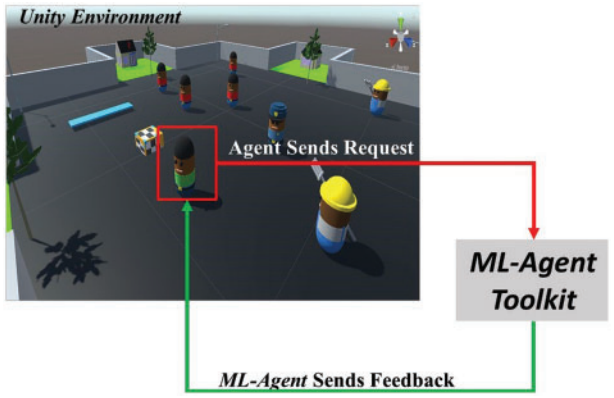


Figure 3: The training of the agent with *ML-Agents*

Table 1: The hyperparameters of PPO + ICM

#	Hyperparameters	Value	#	Hyperparameters	Value
1	Batch size	2048	8	Hidden units	256, 128
2	Buffer size	20480	9	Number of layers	2
3	Learning rate	0.0003	10	Gamma	0.99
4	Beta	0.005	11	Curiosity strength	1.0
5	Epsilon	0.2	12	Maximum steps	5,000,000 2,000,000
6	Lamb	0.95	13	Time of horizon	128
7	Number of epoch	3	14	Normalize	False

The PPO + ICM configuration (hyperparameters), shown in Table 1, has been programmed according to the official documents of ML-Agent [10]. The buffer_size value corresponds to the data size that we need to update our learning model or start the training. The data will be collected from the agent's actions and behavior while the agent is exploring the environment. The data should be twice the batch_size value, which is the number of input data to the neural network for one iteration. For the used buffer_size 10x, the batch_size = 2048. The beta value ensures that the agent explores more new territories during training. The Generalized Advantage Estimate (GAE) [36] employs lambda λ (Lamb in Table 1) to calculate the features of the PPO algorithm. The num_epoch must be between 1 to 3 for building a robust and stable model. Epoch is a set of iterations, and this number can be found using (2). For every 3 epochs, we have 732 iterations ($3 \times 244 = 732$). Every 3 epochs, we will have an update for our model data. For 2 million cases, the model will be refreshed every 98 iterations.

$$\frac{Iteration}{1\ Epoch} = \frac{Max\ Steps}{Buffer\ Size} = \frac{5,000,000}{20480} = 244 \tag{2}$$

In Table 1, the hidden_units value is the number of units in every fully connected network. The num_layers refer to the number of hidden layers after the input layers. The maximum value of layers is 3, and we used 2 in this work for all four experiments. For this work, we choose training our models for 2 million steps as a starting point. The curiosity strength value represents the reward value that will be given to the agent to inspire it to explore and conduct an additional search. The max_steps value is the maximum number of iterations, and we specified and used 2 million and 5 million iterations in our four experiments. For Exp_2, Exp_3, and Exp_4, we ran the simulations with the same hyperparameters as in Table 1, but we changed the number of hidden units and the number of max_steps (iterations) as shown in Table 2. We now have four different models for the four different experiments, as in Table 2, and we will test them in the next section.

Table 2: The configurations of the four experiences

Experiment	Num_hidden_unit	Max_steps
Exp_1	128	2 million
Exp_2	128	5 million
Exp_3	256	2 million
Exp_4	256	5 million

3.4 PPO + ICM Model

A proximal policy optimizer is the practical solution for our case, according to previous research [18,26,27]. PPO is used to train an autonomous agent to explore, search, and train another agent in a real-time environment. The PPO algorithm loss function is continuously minimized using the mathematical model in (3) and is used to update the learning value [11].

$$L_{CLIP}(\vartheta) = \hat{E}t\left[min\left(rt(\vartheta)\hat{A}t, clip(rt(\vartheta),\ 1-\varepsilon, 1+\varepsilon)\hat{A}t\right)\right] \tag{3}$$

where the superscript *CLIP* refers to conservative policy iteration. ϑ is the policy parameter, $\hat{E}t$ is the expectation over time steps, rt is the probability ratio of new and old policies, $\hat{A}t$ is the estimated advantage at time t, and ε is the hyperparameter. Eq. (3) is used to update the learning knowledge of the model with every specific number of iterations. In this work, we have four models, each with a different number of hidden units and a different number of steps (iterations) as described in Table 2. The agent is provided with Raycasts, which is a sensor that measures the distance between the agent and the other obstacles, and Raycasts generates data a digital value, which is stored in a matrix or vector. Based on this data, PPO + ICM algorithm will be trained on this vector, and the model will learn how to direct the intelligent agent without causing any conflict. PPO + ICM algorithm is constructed using Tensorflow/Keras backend neural networks in which ICM uses a forward model and an inverse model to encode states as input and predict both the action and the next state. The differences between the encoded next states and the predicted ones are used as intrinsic reward signals that encourage agents to explore new states, such that the agent policies maximize the expected sum of extrinsic and intrinsic rewards, as shown in Fig. 4, which reviews how the PPO + ICM algorithms work.

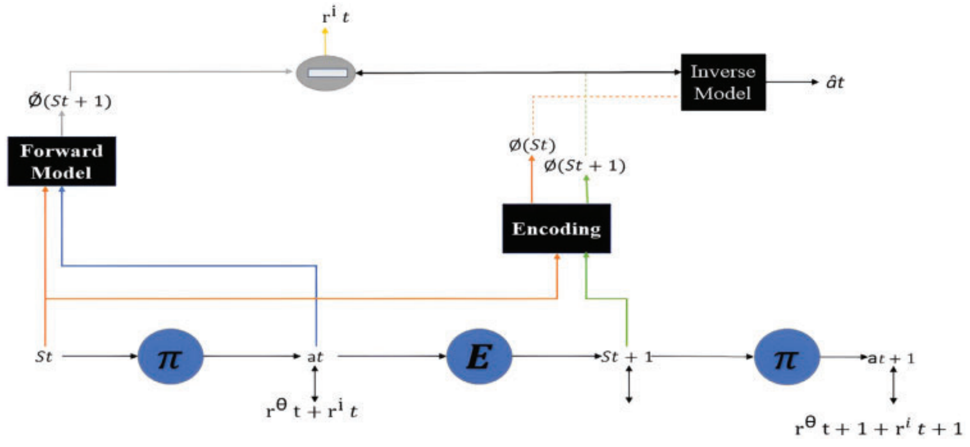


Figure 4: ICM computes the difference of encoded next states and encoded predicted next states

4 Results

Table 2 presents the four implemented experiments with different steps and hidden units. This work involves the construction of four different models for autonomous agents, and we have evaluated these to identify the most effective model for this research. The four models have been designed to guide an agent in a crowded environment surrounded by walls, and we ran the agents in the environment for thirty minutes to evaluate which model was most efficient. Fig. 5 demonstrates how the agent found the target and delivered it to the goal point without accidents. The agent training to search for a specific target. After finding it, he will attempt to drive it to the goal with a cyan color in Fig. 5. While the agent tries to reach the target or the goal, he will try to evade any obstacles in his path. The challenge for our agent is reaching the goal with no or with minimal collisions.

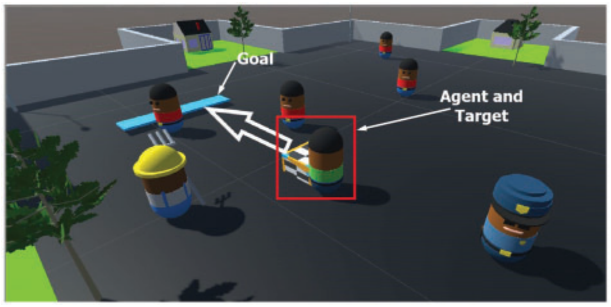


Figure 5: The agent drives the target to the goal

Fig. 6 demonstrates the GC-Ratio, the number of collisions gained during training, the number of episodes, which is the number of rounds, and a number of goals which is the number of successful attempts that the agent reached the goal during training.

Fig. 7 presents the agent's performance in the *Unity* environment. We can see that the episode or the behavior time is decreasing, which shows that the agent learns to drive the target to the goal with fewer movements in the following iterations. The cumulative reward was given to the agent, so it guarantees that the future reward will be bigger to motivate and encourage the agent to explore more new areas.

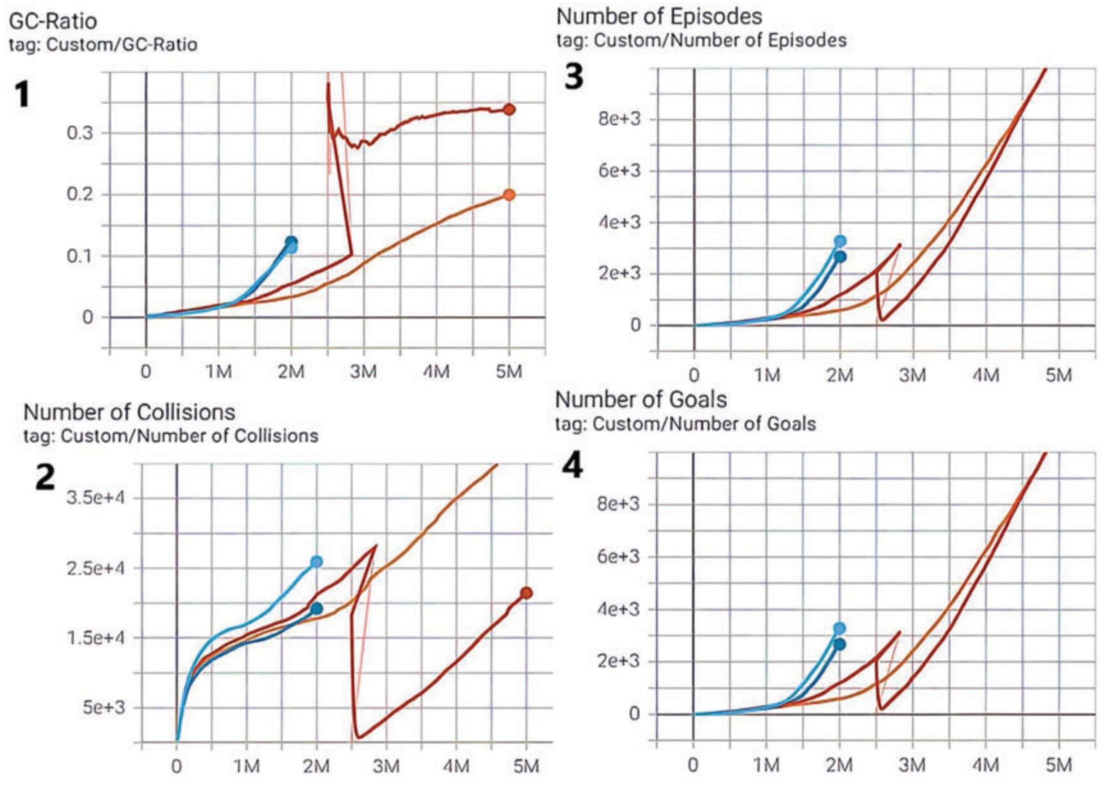


Figure 6: Four curves are presented, each with different colored lines, representing the model's performance for each of the four experiments. The cyan line represents Exp_1. The red line represents Exp_2. The blue line represents Exp_3. The orange line represents Exp_4. The agent's performance curves: (1) GC-Ratio calculates how much the gained ratio during training; (2) the number of collisions value gained during training; (3) the number of episodes, which is the number of rounds; (4) the number of goals is the number of the successful attempts that the agent reached the goal during training

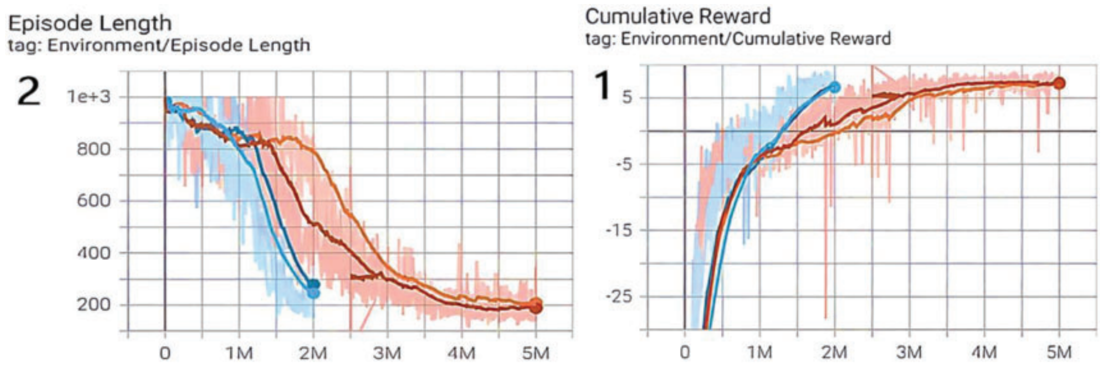


Figure 7: The training and reward processes: (1) Cumulative reward curve. (2) Episode length: Episode means the action or behavior of the agent, and the length of the episode means the time the agent took to reach the goal in every iteration during training

Fig. 8 illustrates the value and policy losses for the designed model. The value loss is the difference between the predicted and actual values. It is used to determine the predictable reward for an agent and to update the parameters of the value function network. The policy loss is the difference between the current and previous policies. Every environment has a special policy or environment rules, which the agent needs to learn to succeed in this environment. The policy loss is used to update the parameters of the policy network to enhance the model's performance.

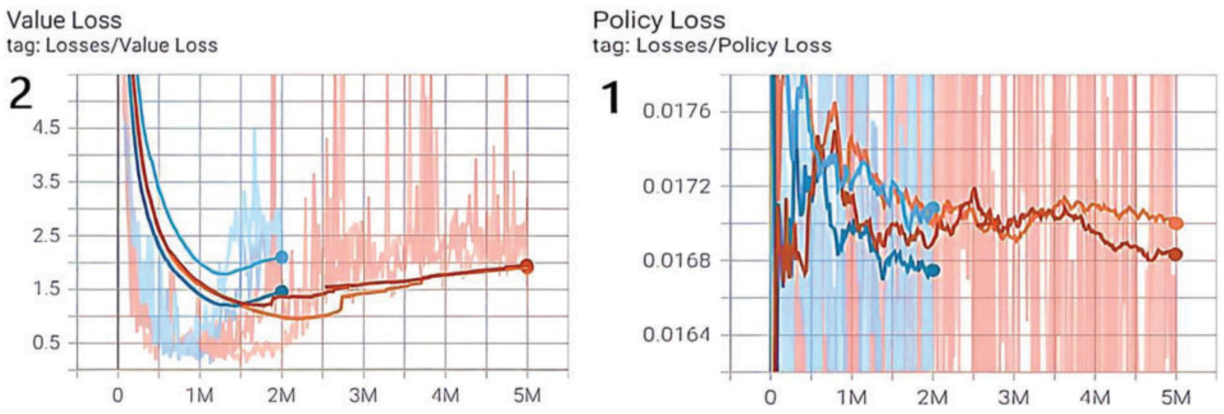
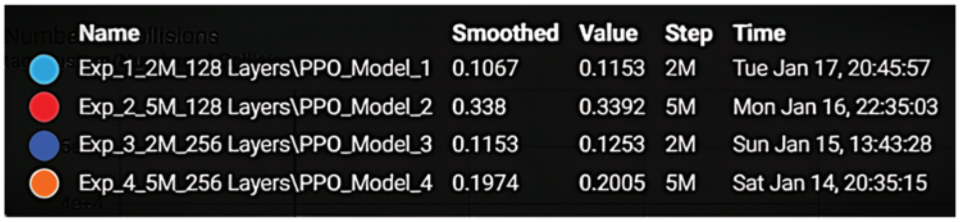


Figure 8: The loss reduction: (1) The policy loss is the difference between the current and previous policies. (2) The value loss is the difference between the predicted and actual values

We tested the four models we designed for the autonomous agent in a *Unity* environment for 30 min. Fig. 9 is a snapshot from the training that illustrates the simulation summary of the training models. The colored circles refer to the experiment curve color. *Smoothed* is the smoothness ratio in the plotted curves. *Step* is the number of iterations. The experiment "Name" and the color of each curve for every model are: Exp_1_2M_128 layers (cyan), Exp_2_5M_128 layers (red), Exp_3_2M_256 layers (blue), and Exp_4_5M_256 layers (orange). In the next section, we will explain the differences between models, and select the model that provides the best performance.



	Name	Smoothed	Value	Step	Time
●	Exp_1_2M_128 Layers\PPO_Model_1	0.1067	0.1153	2M	Tue Jan 17, 20:45:57
●	Exp_2_5M_128 Layers\PPO_Model_2	0.338	0.3392	5M	Mon Jan 16, 22:35:03
●	Exp_3_2M_256 Layers\PPO_Model_3	0.1153	0.1253	2M	Sun Jan 15, 13:43:28
●	Exp_4_5M_256 Layers\PPO_Model_4	0.1974	0.2005	5M	Sat Jan 14, 20:35:15

Figure 9: Snapshot for summary of the training models. The colored circles refer to the experiment curve color. *Smoothed* is the smoothness ratio in the plotted curves. *Step* is the number of iterations

4.1 Comparison

A comparison was made between the four implemented models in the *Unity* environment. The evaluation for every model has been done according to the following four metrics:

- **Goal:** The number of successful attempts that the agent reached the goal in 30 min.
- **Collision:** The number of failed attempts in which the agent made a mistake or an accident by hitting other characters or walls.

- **GC-Ratio:** The ratio for the goal collision calculated using (1), which we presented in Section 3.
- **Rounds:** As presented in Table 3, the total number of rounds equals the number of goals in 30 min.

Table 3: The performance of the models

Experiment	Goal	Collisions	GC-ratio	Rounds	Model performance
Exp_1	99	81	0.550997	99	2nd Worst
Exp_2	94	134	0.413278	94	Worst
Exp_3	101	57	0.640236	101	Best
Exp_4	73	52	0.599355	73	2nd Best

Table 3 presents the performance results for the four models within a 30-min simulation window, where Exp_3 generated the best performance. The agent scored 101 goals, which is the highest number of goals in this work with the highest value of GC-Ratio of 0.64. The agent had 57 collisions in 101 rounds, which is the highest number of rounds in the four experiments.

5 Discussion

This work was not expected to totally prevent collisions; in fact, it was possible as expected to reduce the collisions and enable the intelligent agent to avoid the blocks and obstacles while attempting to locate the target or direct the target to the goal. An additional limitation is related to when the autonomous agent delivers the goal to the target. Here, the agent stops avoiding the other characters and obstacles. This issue is associated with the limitation of the agent and the environment. If the agent could transport the target, this would solve the problem, and our agent can turn his attention to his path instead of pushing the target.

There were two questions we raised in the research:

Q1: Are goals and collisions good metrics for choosing a model of an autonomous explorer agent?

If we want to utilize a good model, we need a model with a larger number of goals and a smaller number of collisions. The GC-Ratio is a good metric for choosing the model. The GC-Ratio automatically computes the difference between goals and collisions, like in our Exp_3 and Exp_4. Exp_3 generated a GC-Ratio value of 0.64, which is greater than the Exp_4 GC-Ratio value of 0.599. Therefore, Exp_3 is considered the best and most robust model among the four models we implemented.

Q2: Are 256 hidden units the best to minimize the number of collisions for an autonomous explorer agent to avoid the negative side effects?

Four models were implemented, each generating a different result, as described in Table 3. Our results show that Exp_3 and Exp_4 were the best and second-best models, respectively. In Exp_3 and Exp_4, we could reduce the number of collisions to less than those obtained in Exp_1 and Exp_2. In this case, the experiments with the 256 hidden units could minimize the number of collisions and provide us with stable and high-quality models that work better than the models with 128 hidden units.

6 Conclusion and Future Work

This paper describes four models that can enable an autonomous agent to explore and navigate in a crowded environment with less harm by reducing the number of collisions. We trained each model with a different configuration according to the four experiments we designed. We equipped our intelligent agent with individual models to measure how well each would function in a crowded *Unity* environment. During the thirty-minute simulation window, our evaluation of the four models revealed that the third model was most successful in directing the agent to a secure and proper path. Using reinforcement learning and adding 256 hidden units enabled to design of a reliable algorithm that can direct the intelligent agent to perform the required task. The agent achieved the highest number of goals in this work, which is 101 and had a GC-Ratio of 0.64 as the highest value. During the 101 rounds of the four experiments, the agent had the smallest number of collisions at 57. The agent avoided obstacles while searching for the target and delivered it to the goal point. Implementing reinforcement learning to supervise the behavior of an intelligent agent generated a self-learning AI safety model and facilitates the way to design a control system with less effort, without guidance, and without the requirement for a mathematical model. Future work is planned to be done using the Yolo7 algorithm for object detection and we will investigate the possibility of employing the segmentation method to secure safe navigation.

Acknowledgement: The authors acknowledge the contribution and the support of the Department of Electrical & Computer Engineering and Computer Science at Jackson State University (JSU).

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: A. Abdulghani, M. Abdulghani, W. Walters, K. Abed; data collection: A. Abdulghani; analysis and interpretation of results: A. Abdulghani, M. Abdulghani, W. Walters, K. Abed; draft manuscript preparation: A. Abdulghani, M. Abdulghani, K. Abed. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The code used and/or analyzed during this research are available from the corresponding author upon reasonable request.

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] J. Garćıa and F. Ferńandez, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, pp. 1437–1480, 2015.

[2] J. Hu, X. Yang, W. Wang, P. Wei, L. Ying *et al.,* "Obstacle avoidance for UAS in continuous action space using deep reinforcement learning," *IEEE Access*, vol. 10, pp. 90623–90634, 2022. https://doi.org/10.1109/ACCESS.2022.3201962

[3] P. Mallozzi, R. Pardo, V. Duplessis, P. Pelliccione and G. Schneider, "MoVEMo: A structured approach for engineering reward functions," in *Second IEEE Int. Conf. on Robotic Computing (IRC)*, Laguna Hills, CA, USA, pp. 250–257, 2018.

[4] M. J. Mataric, “Reward functions for accelerated learning,” in *Machine Learning: Proc. of the Eleventh Int. Conf.*, New Brunswick, USA, pp. 181–189, 1994.

[5] M. M. Abdulghani and K. M. Al-Aubidy, “Design and evaluation of a MIMO ANFIS using MATLAB and VREP,” in *11th Int. Conf. on Advances in Computing, Control, and Telecommunication Technologies, ACT 2020*, 2020.

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

[7] 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

[8] C. Rosser and K. Abed, “Curiosity-driven reinforced learning of undesired actions in autonomous intelligent agents,” in *2021 IEEE 19th World Symp. on Applied Machine Intelligence and Informatics (SAMI)*, Herl’any, Slovakia, pp. 000039–000042, 2021.

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

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

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

[12] D. Silver, J. Schrittwieser, K. Simonyan and I. Antonoglou, “Mastering the game of go without human knowledge,” *Nature*, vol. 550, no. 7676, pp. 354, 2017.

[13] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets *et al.,* “StarCraft II: A new challenge for reinforcement learning,” arXiv:1708.04782, 2017.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu and J. Veness, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529, 2015.

[15] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski *et al.,* “Rainbow: Combining improvements in deep reinforcement learning,” in *Proc. 32nd AAAI Conf. Artif. Intell.*, New Orleans, Louisiana, USA, pp. 3215–3222, 2018.

[16] D. Weld and O. Etzioni, “The first law of robotics,” In: M. Barley, H. Mouratidis, A. Unruh, D. Spears, P. Scerri, F. Massacci (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

[17] Y. Chow, O. Nachum, E. Duéñez-Guzmán and M. Ghavamzadeh, “A Lyapunov-based approach to safe reinforcement learning,” *Neural Information Processing Systems*, vol. 31, pp. 908–918, 2018.

[18] 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.

[19] F. Berkenkamp, M. Turchetta, A. P. Schoellig and A. Krause, “Safe model-based reinforcement learning with stability guarantees,” arXiv:1705.08551, 2017.

[20] Y. Chow, M. Ghavamzadeh, L. Janson and M. Pavone, “Risk-constrained reinforcement learning with percentile risk criteria,” arXiv:1512.01629, 2015.

[21] C. J. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 1992, pp. 279–292, 1992.

[22] P. Alamdari, T. Klassen, R. Icarte, S. Icarte and S. A. McIlraith, “Be considerate: Avoiding negative side effects in reinforcement learning,” in *Proc. of the 21st Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2022),* Online, pp. 9, 2022.

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

[24] B. Eysenbach, S. Gu, J. Ibarz and S. Levine, “Leave no trace: Learning to reset for safe and autonomous reinforcement learning,” in *Int. Conf. on Learning Representations*, Vancouver, BC, Canada, 2018.

[25] A. Hans, D. Schneegaß, A. M. Schäfer and S. Udluft, “Safe exploration for reinforcement learning,” *The European Symp. on Artificial Neural Networks*, Bruges, Belgium, 2008.

[26] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell *et al.,* "Emergent tool use from multi-agent autocurricula," arXiv:1909.07528, 2020.

[27] P. Reizinger and M. Szemenyei, "Attention-based curiosity-driven exploration in deep reinforcement learning," in *ICASSP 2020-2020 IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, pp. 3542–3546, 2020.

[28] V. Goecks, G. Gremillion, V. Lawhern, J. Valasek and N. Waytowich, "Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real-time," arXiv:1810:11545, 2019.

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

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

[31] B. Li, T. Lu, J. Li, N. Lu, Y. Cai *et al.,* "Curiosity-driven exploration for off-policy reinforcement learning methods," in *2019 IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, Dali, China, pp. 1109–1114, 2019.

[32] T. Ward, A. Bolt, N. Hemmings, S. Carter, M. Sanchez *et al.,* "Using unity to help solve intelligence," arXiv:2011.09294, 2020.

[33] M. Johansen, M. Pichlmair and S. Risi, "Video game description language environment for unity machine learning agents," in *2019 IEEE Conf. on Games (CoG)*, London, UK, pp. 1–8, 2019.https://doi.org/10.1109/CIG.2019.8848072

[34] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury *et al.,* "PyTorch: An imperative style, high-performance deep learning library," *Neural Information Processing Systems*, vol. 32, 2019.

[35] R. S. Dehal, C. Munjal, A. A. Ansari and A. S. Kushwaha, "GPU computing revolution: CUDA," in *2018 Int. Conf. on Advances in Computing, Communication Control and Networking (ICACCCN)*, Greater Noida, India, pp. 197–201, 2018. https://doi.org/10.1109/ICACCCN.2018.8748495

[36] J. Schulman, P. Moritz, S. Levine, M. I. Jordan and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," arXiv:1506.02438, 2015. https://doi.org/10.48550/arXiv.1506.02438