iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 10 **Issue:** VII **Month of publication:** July 2022

DOI: https://doi.org/10.22214/ijraset.2022.45526

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VII July 2022- Available at www.ijraset.com

Learning to Survive using Reinforcement Learning with MLAgents

Sarosh Dandoti
Vishwakarma University

Abstract: Simulations have been there for a long time, in different versions and level of complexity. Training a Reinforcement Learning model in a 3D environment lets us understand a lot of new insights from the inference. There have been some examples where the AI learns to Feed Itself, Learns to Start walking, jumping etc. The reason one trains an entire model from the agent knowing nothing to being a perfect task achiever is that during the process, new behavioral patterns can be recorded. Reinforcement Learning is a feedback-based Machine Learning technique in which an agent learns how to behave in a given environment by performing actions and observing the outcomes of those actions. For each positive action, the agent receives positive feedback; for each negative action, the agent receives negative feedback or a penalty. A general simple agent would learn to perform a task and get some reward on accomplishing it. The Agent is also given punishment if it does something that it's not supposed to do. These simple simulations can evolve, try to use their surroundings, try to fight with other agents to accomplish their goal.

Key Words: Unity3d, Simulation, Machine Learning, Programming, Logic, C#, Reinforcement Learning, Neural Networks.

I. INTRODUCTION

In this implementation , Unity 3d software is using to peform the Training. Unity has a MLAgents library which allows us to perform Machine Learning projects in their 3d Environment. The Unity Machine Learning Agents Toolkit, often known as ML-Agents, is an open-source Unity project that enables the use of games and simulations as training grounds for intelligent agents. ML-Agents offers a cutting-edge ML library to train agents for 2D, 3D, and VR/AR environments, as well as a C# software development kit (SDK) to set up a scene and specify the agents within it. With the help of Unity Machine Learning Agents (ML-Agents), developers may produce more engaging games and a better gaming experience. A developer using the platform can use deep reinforcement learning and imitation learning to teach intelligent agents how to learn. In our project we are going to use a simple lion model where it learns to feed itself and its cub and we are going to further makes changes in its environment , introduce competitors, enemies , etc.. This Model will start from scratch knowing nothing initially and will be wondering randomly in the initial states of learning iterations. Overtime it will learn to eat chickens , feed it to its cub and more further actions to improve its survival. We will try different instances of these training simulations to see get a simple inference of how and if the model evolves with new forces that come into play in each in project. There's a few components and terminology we need to understand before we move forward.

Agent(): A thing that can see and investigate its surroundings and take appropriate action.

Environment(): The context in which an agent exists or is created. In RL, we take the assumption that the environment is stochastic, or essentially random.

Action(): An agent's movements inside the environment are referred to as actions.

State(): State is a situation that the environment returns following each action the agent takes.

Reward(): An evaluation of the agent's performance based on feedback received from the environment..

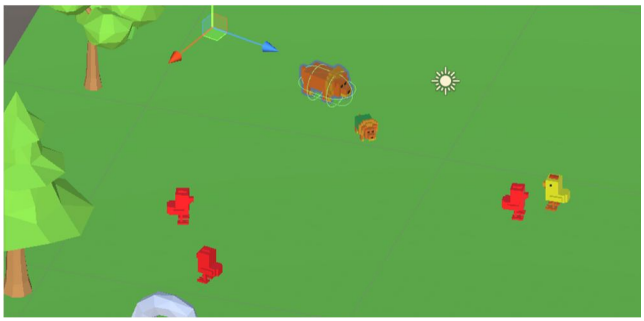
For our reinforcement learning example, the training phase learns the optimal policy through guided trials, and in the inference phase, the agent observes and tales actions in the wild using its learned policy.

II. GOAL



©IJRASET: All Rights are Reserved

The main aim here is to train our model to feed its cub/ itself the chickens, or just good chickens which give reward. This can be generalized by another simpler example of a man collecting berries from a forrest and storing it in a granary. The examples are limitless in this instance of performing a task, avoiding some things, collecting some things, and running away from some other objects. We will also have multiple agents further down the project where the agents will fight each other to steal their chickens and give it to their cubs.



III. SETTING UP THE ENVIRONMENT

Creating and setting up a proper environment for training our agent is one of the most important tasks while doing any ML project in unity. We don't want our agent to fall off the map or jump over some things unnecessarily. So we create a simple map where we have land. The Lion, cubs and chickens can spawn anywhere on the map.

In there first project we will have only one type of chicken, where the goal is to take the chicken and feed it to the cub.

Let's define our conditions:

The lion can move left , right and forward only , its action parameters would be , turn-left, turn-right , dont_turn-left , dont_turn-right, move-forward, stay.

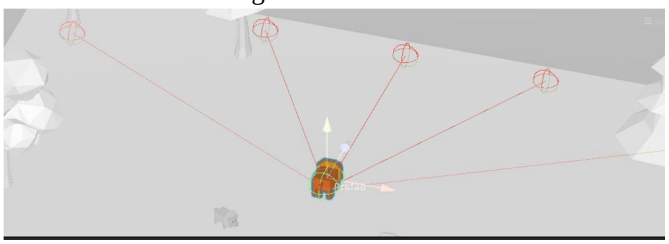
The lion will get a reward for picking up a Green chicken.

The lion will die if it picks up a red chicken and we minus the reward. Then the scene will be reset.

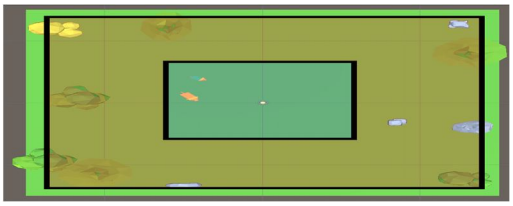
If after a certain time the lion is unable to find food/any reward, we minus the reward and reset the scene, So the lion is motivated to move faster in search of a reward.

Feeding the cub while the chicken is picked up also gives reward. Collision with the cub with no food results in nothing.

The lion agent observes the environment in two different ways. The first way is with raycasts. This is like shining a bunch of laser pointers out from the penguin and seeing if they hit anything. It's similar to LIDAR, which is used by autonomous cars and robots. Raycast observations are added via a RayPerceptionSensor component. The second way the agent observes the environment is with numerical values. Whether it's a true/false value, a distance, an XYZ position in space, or a rotation, you can convert an observation into a list of numbers and add it as an observation for the agent.



You must choose your subjects for observation with great care. The agent won't be able to finish its duty if it doesn't have enough knowledge of its surroundings. Imagine your agent is blindfolded and floating in outer space. What information about its surroundings would be necessary for it to make wise decisions?



ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VII July 2022- Available at www.ijraset.com

The reddish area on the outer portion of the square is where an enemy will spawn in the later half of this project.
The Lion Agent and the Cub will spawn anywhere within the smaller green square. The chickens/food will spawn anywhere on the map.

IV. TRAINING

In unity 3d , Training works in Episodes , everytime the timer to do something runs out or if the agent does something which will punish it , the episode resets and we reset our environment. For example , we the lion collides with the red chicken , we add a negative reward to it , end the episode, reset the entire environment and start a new episode again.
Implement the OnEpisodeBegin() method to create an Agent instance at the start of an episode.
EndEpisode() resets the agent and sets the done flag to true.
We collect all the information from the agent model through the CollectObservations method provided by the ML-Agents Package .We use CollectObservations() to gather the agent's vector observations for the step.

```
public override void CollectObservations(VectorSensor sensor)
{
    // Whether the lion has eaten a Chicken (1 float = 1 value)
    sensor.AddObservation(isFull);

    // Distance to the cub (1 float = 1 value)
    sensor.AddObservation(Vector3.Distance(cub.transform.position, transform.position));

    // Direction to cub (1 Vector3 = 3 values)
    sensor.AddObservation((cub.transform.position - transform.position).normalized);

    // Direction lion  is facing (1 Vector3 = 3 values)
    sensor.AddObservation(transform.forward);

    // 1 + 1 + 3 + 3 = 8 total values
}
```

The First Observation is isFull to check whether the lion has eaten. You can pass different data type variables in this AddObservation(). The Next Observation is distance to the cub. Keep in mind this has to be the relative distance from the lion agent to the cub, hence it is
Vector3.Distance(cub.transform.position, transform.position)
The Next Observation is direction to he cub,
(cub.transform.position - transform.position).normalized
Then we have the lion facing observation, which basically observes the direction the agent is facing while all the data is being recorded/observed.
From the agent's point of view, the agent observation describes the present environment. Any information about the environment that aids an agent in achieving its objective is considered an observation. For a fighting agent, for instance, its observation might include the distances to allies or adversaries, or the quantity of ammunition it currently has available.
We need to also provide parameters for our Neural Network and they are easy to tune. We all the parameters like batch_size , learning_rate , epochs , layers m etc , for the neural network in a .yml file. We have tried to explain some of the parameters in our project. An important parameter while training in Unity is MAX_STEPS - Total number of actions that must be completed in the environment (or across all environments if employing multiple in parallel) before the training process is concluded .If our environment has many agents with the same behaviour name, all of those agents' steps will add up to the same max steps count.
In this project we are using the trainer type : ppo.
Proximal Policy Optimization is a reinforcement learning method used by ML-Agents (PPO). In PPO, the optimal function that connects an agent's observations to the best course of action in a given state is approximated using a neural network.
batch_size : refers to the number of training examples utilized in one iteration
buffer_size corresponds to how many experiences (agent observations, actions and rewards obtained) should be collected before we do any learning or updating of the model. This should be a multiple of batch_size. Typically larger buffer_size correspond to more stable training updates. *Typical Range: 2048 - 409600*
time_horizon : corresponds to how many steps of experience to collect per-agent before adding it to the experience buffer. When this limit is reached before the end of an episode, a value estimate is used to predict the overall expected reward from the agent's current state. In cases where there are frequent rewards within an episode, or episodes are prohibitively large, a smaller number can be more ideal. This number should be large enough to capture all the important behavior within a sequence of an agent's action.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

num_layers corresponds to how many hidden layers are present after the observation input, or after the CNN encoding of the visual observation. For simple problems, fewer layers are likely to train faster and more efficiently. More layers may be necessary for more complex control problems. *Typical range: 1 - 3*

hidden_units correspond to how many units are in each fully connected layer of the neural network. For simple problems where the correct action is a straightforward combination of the observation inputs, this should be small. For problems where the action is a very complex interaction between the observation variables, this should be larger. *Typical Range: 32 - 512*

```
behaviors:
    Lion:
        trainer_type: ppo
        hyperparameters:
            batch_size: 128
            buffer_size: 2048
            learning_rate: 0.0003
            beta: 0.01
            epsilon: 0.2
            lambd: 0.95
            num_epoch: 3
            learning_rate_schedule: linear
        network_settings:
            normalize: false
            hidden_units: 256
            num_layers: 2
            vis_encode_type: simple
        reward_signals:
            extrinsic:
                gamma: 0.99
                strength: 1.0
        keep_checkpoints: 5
        max_steps: 1000000
        time_horizon: 128
        summary_freq: 5000
        threaded: true
```

While training is being performed we can see the penguin move randomly and it does not have any sense of direction of knowledge. But after sometime we observe that the that penguin learns to pick up fishes, intitally both, red and green, it resets if it picks up red but after some more training we observe that the agent easily avoids the red fish and only picks up the green fishes. Hence it has achieved its purpose successfully.

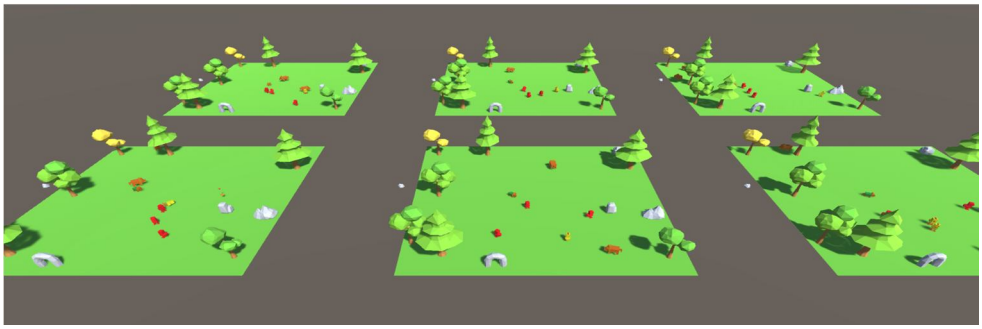
V. INFERENCE

The training can be stopped after a while when the Mean Reward Value start to stabilize. Unity automatically saves this into a model with a .onnx file extension and can be imported back into unity for more tests. ONNX (Open Neural Network Exchange) is an open format for ML models. It allows you to easily interchange models between various ML frameworks and tools.

The model after a number of steps gets very good at finding the green chicken and feeding it to the cubs, it also very conveniently just moves by the red chickens and does not collide with it. However once the green chickens are depleted the lion model just randomly moves around the environment searching for more work that rewards it, but does not in any case eat the red chicken.

Now this model would be working on its own and its gives us freedom to test another model and see how these both models react to each other. This now helps to create another similar project in the same environment but with a new agent now in play.

It is worth mentioning that if we provide two lion agent models and only one cub for this scenario, both lions will feed the cub.

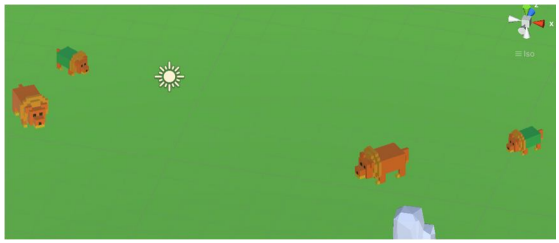


ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VII July 2022- Available at www.ijraset.com

VI. INTRODUCING A RIVAL AGENT

It is only natural that living beings go to extreme measures to get food and feed themselves and their family. Hence in this version of the project , a new lion and a new cub is introduced where lion_1 tries to steal the food by colliding into the lion_2 . This does seem very basic but we are trying to test variations and see if there's a significant outcome.

If a lion agent has food , its tag in unity is set to a different one (hasFood) but a lion with no food is just with tag (Lion) , This allows the other lion agent to see through the Raycast sensors if the opponent has food or not. If the lion collide with the other lion agent while it has food , the food item it transferred to the first lion and is rewarded for it , meanwhile, the second lion is given a small negative reward. This give the agents a sense of decision making of whether it is worth it to collide with other lions with food or not. If a lion agent has food, it will then obviously try to avoid other lions since its food / chicken can be stolen . But this lets the agent vulnerable to other lions. So this creates a decision, whether a lion should find a chicken or steal it from other lions.

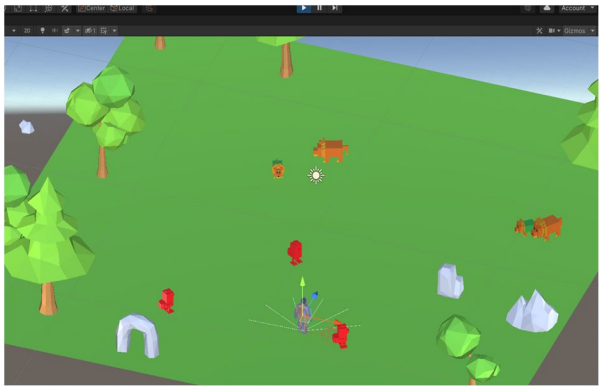


After the training on the newer model is done , we replace both agents models with the new saved model, so that both lions can now have the option to steal.

In many cases, lions first look for chickens and during the last one remaining chicken they resort to stealing , which by nature only seems logical. Here's how it was trained. The original first model was used and a new agent which is now learning had the stealing part of code written in it. Now only the new lion agent would learn to steal while the other lion, if its food was stolen , would just go back to finding another chicken.

VII. INTRODUCING AN ENEMY

In a real life environment, theres always a bigger fish, or no one is the biggest predator, in this new variant of the project , we remove the extra lion and keep only one of the lions and then retrain it again with some changes. We take a human 3D model and take it as the enemy.



Conditions of the Enemy:

This enemy will be randomly spawned a bit away from the lion agent.

It will roam randomly in the given environment just like the chickens.

It will not have any interaction with the cub or the chickens.

It will have a small field of view from where it can see the lion using a raycast sensor.

The lion will get a very small amount of negative reward if it comes in view of the enemy and every time the lion gets closer to the enemy the negative reward is slightly increased, until it finally touches the enemy and dies , where we give a bigger punishment and then reset the environment.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

This model resulted in excellent control of its motion, over time it clearly learned how to avoid areas with the enemy present. It is worth noting that if there was no alternative route to a food , the agent did enter the field of view of the enemy as the reward for the chicken beyond the field was much bigger than the punishment for being in the field (enemy zone).

VIII. FUTURE

We are planning to create this project as an open source simulation platform with various other animals in the environment to see how they would behave and interact with each other in order to ensure the survival of them and their offsprings. There is a huge potential in trying to recreate nature and it helps to various behavioural patterns in life. It is challenging for analysts or engineers to apply reinforcement learning widely because it needs a lot of data to operate accurately and effectively. As a result, the following situations can benefit from reinforcement learning: When interacting with an environment, the agent doesn't need to do arbitrary activities because the environment can be accurately represented. Because trained agents may act without knowing the system's future state in advance, they are more successful than unsupervised ML algorithms when there is not enough real-time data to make decisions (environment).

RL has been extensively employed in fields like robotics and computer games, among others. The most intriguing feature of RL is its capacity for problem solving in the absence of any prior knowledge of the environment or state. The key benefit of adopting this strategy is that it can solve complex nonlinear functions that are challenging to fit using other machine learning techniques.

IX. CONCLUSION

This project works correctly and is tested. Creating and observing simulations and deriving insights and understanding life from them has always been a very interesting field of learning. Reinforcement Learning has a variety of applications and this specific project could be used in larger life recreation simulation systems , videogame AIs , etc. Robotics for industrial automation. Business strategy planning, It helps you to create training systems that provide custom instruction and materials according to the requirement of students. Aircraft control and robot motion control

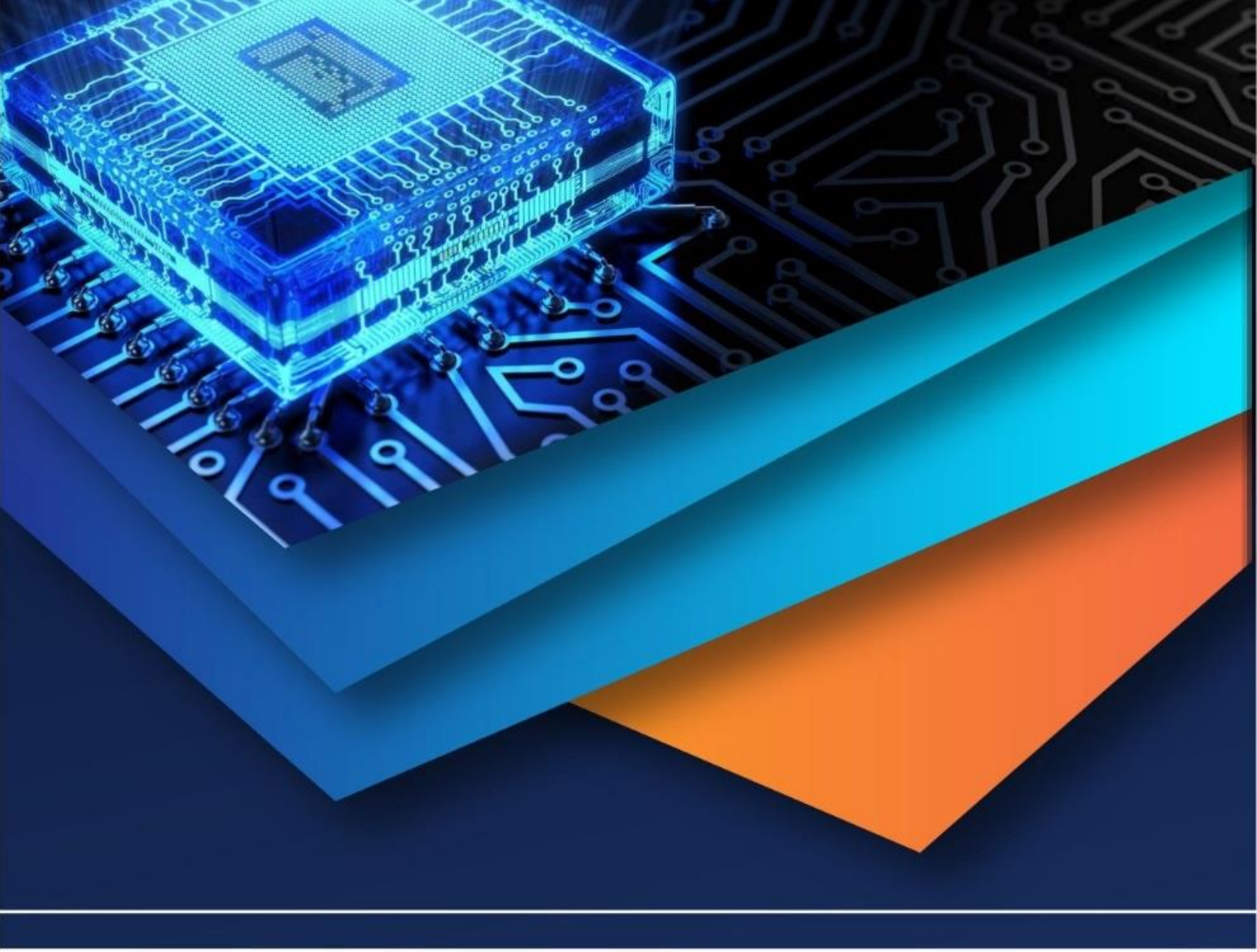
REFERENCES

[1] Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D. (2020). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents.

[2] Nandy, A., Biswas, M. (2018). Unity ML-Agents. In: Neural Networks in Unity. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-3673-4_2

[3] A. E. Youssef, S. E. Missiry, I. Nabil El-gaafary, J. S. ElMosalami, K. M. Awad and K. Yasser, "Building your kingdom Imitation Learning for a Custom Gameplay Using Unity ML-agents," 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2019, pp. 0509-0514, doi: 10.1109/IEMCON.2019.8936134

[4] Majumder, A. (2021). Setting Up ML Agents Toolkit. In: Deep Reinforcement Learning in Unity. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6503-1_3



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)