**Application of Gaming Using Augmented Reality and Reinforcement Learning**

**Submitted for**

**INTELLIGENT MODEL DESIGN  
 USING AI**

Submitted by:

**(E23MCAG0065) Kondepudy Karthikeya**

Submitted to

**DR. NITIN ARVIND SHELKE**

**Jan-May 2024**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

A close-up of a logo

Description automatically generated

Table of Contents

Table of Contents ii

Revision History ii

1. Introduction 1

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Project Scope 1

1.5 References 1

2. Overall Description 2

2.1 Product Perspective 2

2.2 Product Features 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 2

2.6 User Documentation 2

2.7 Assumptions and Dependencies 3

3. System Features 3

3.1 Social Interaction 3

3.2 4

4. External Interface Requirements 4

4.1 User Interfaces 4

4.2 Hardware Interfaces 4

4.3 Software Interfaces 4

4.4 Communications Interfaces 4

5. Other Nonfunctional Requirements 5

5.1 Performance Requirements 5

5.2 Safety Requirements 5

5.3 Security Requirements 5

5.4 Software Quality Attributes 5

6. Other Requirements 5

Appendix A: Glossary 5

Appendix B: Analysis Models 6

Appendix C: Issues List 6

Application of gaming using Augmented Reality (AR) and Reinforcement Learning

Kondepudy Karthikeya

School of Computer Science and Engineering Technology, Bennett University

**Abstract**

We aim to develop an Augmented Reality (AR) game using the Unity Game Engine and an in-built package called AR Foundation Kit, our focus will be to create a wildlife simulator. Making use of the ARCore (By Google) and ARKit (By Apple) tools, our game will blend virtual (3d modelled) animals with the player's real-life environment, our objective is to provide an immersive and captivating gameplay experience. Another key point in our project is the integration of machine learning (ML) techniques, specifically reinforcement learning, to govern the behaviours and animations of our virtual animals. We make use of the ML Agents package, ensuring that animal behaviours are realistic, enhancing the experience of the users. Our coding environment will be Visual Studio using the C# programming language, taking full advantage of the pre-integrated environment in Visual Studio for Unity. Our project addresses the lack of hit/successful AR games on smartphone platforms, recognizing the vast potential for innovation in this field. Our app will offer support for Android 7.0 (Nougat) and iOS 11.0 or later versions, the gameplay will feature one animal: Penguins. Future updates will introduce more animals, further enriching the player experience and expanding the possibilities within the game.

**Keywords:** Augmented Reality,Unity, Machine Learning, Android, iOS

1. **Introduction**

In recent years, the convergence of augmented reality (AR) technology and gaming has ushered in a new era of interactive entertainment, blurring the lines between the virtual and physical worlds. With the widespread adoption of smartphones equipped with powerful sensors and cameras, AR games have captivated audiences worldwide, offering immersive experiences that transcend traditional gaming boundaries. In this dynamic landscape, our project emerges as a pioneering endeavour to push the boundaries of AR gaming, combining the allure of wildlife simulation with the power of machine learning (ML) to create an unparalleled gaming experience. The fusion of AR technology with wildlife simulation opens exciting possibilities for gamers to engage with virtual creatures in their real-world surroundings. By seamlessly overlaying digital assets onto the physical environment, players are transported into a realm where virtual animals roam freely, exhibiting behaviours that mimic their real-life counterparts. However, while existing AR games have showcased the potential of this concept, few have ventured into the realm of ML-driven animal behaviours, limiting the depth and authenticity of the gameplay experience. Our project seeks to address this gap by integrating ML techniques, including reinforcement and imitation learning, to govern the behaviours of virtual animals within the AR environment. Leveraging the Unity Game

Engine and the AR Foundation Kit, we aim to create a wildlife simulator that not only captivates players with stunning visual fidelity but also immerses them in a dynamic ecosystem where every interaction feels authentic and meaningful. Through meticulous development and innovative design, our project aspires to redefine the possibilities of AR gaming, offering players a truly immersive and engaging experience that blurs the boundaries between fantasy and reality. By embracing the latest advancements in AR technology and ML algorithms, we envision a future where players can explore and interact with virtual worlds in ways never imagined, ushering in a new era of gaming innovation and creativity.

1. **Background And Motivation**

AR stands for Augmented reality where the user’s present reality/environment is used, and virtual elements are added to it. AR reality provides an improved method for creating, organising, and providing easily understandable instructions by superimposing digital content onto real-world work surroundings. Also employed in this scenario is Reinforcement Learning (RL), which is a Machine Learning (ML) technique whose main objective is to train software (or AI) to make decisions to achieve the most optimal results. It takes after the trial-and-error method that us humans use in our daily lives. For us as computer science students who are also video game fanatics this project is the perfect way to learn about the field, which we are immersing ourselves in and try adding in video gaming elements as well.

2.1 Related Work

Looking back there have been previous works regarding the integration of AR environments with ML elements.[1] For example, *et al. Chengxi Li* Introduces an AR-assisted system architecture for mutual-cognitive safe human robot interaction, where they introduced a curriculum learning-based policy for collision avoidance, with each lesson in the curriculum being more difficult than the last, helping the model train more effectively.

Another example of a similar type of integration which is more closely related to our own use case is,[2] *et al. Robert Huy Le* ‘s work, where they have made use of the YOLOv3 (You Only Look Once, Version 3) object detection algorithm and the AR development toolkit ARKit provided by Apple. Their main aim was to improve the image processing capabilities of AR applications using intelligent machine learning models with help of YOLOv3. Interesting to note is that our unity environment will also be making use of ARKit as well.

Shifting our attention to another implementation of AR and ML by,[3] *et al Ghina Dandachi* where they try to provide us with an improved approach for image augmentation in an AR environment, by acting on two axes in the AR process. They use an ML step for the detection part, then they register the augmented image by processing it using various techniques like statistical appearance models, and covariance matrices of dense image descriptors.

We see [4]*et al Volodymyr Minh* training Deep Reinforcement Learning to train their agent to play the game known as Atari. They do this by learning to control policies directly from HD sensory input. There is a CNN (Convoluted Neural Network) that is trained using a variant of Q-learning where raw pixels act as input and the output is a value function estimating future rewards. They tested this on 6-7 games and obtained state-of-the-art results for their models, which motivated us to try our hand in this field.

There is [5] *et al Haddo Van Hasselt* using

Deep RL, by making use of the Double Q Learning, here they are proposing a specific adaptation to the DQN algorithm and show that it reduces the estimations observed by significant margin. Consequently, leading to better performance as well. [6] *Et al Matteo Hessel* also make use of the DQN algorithm by making full use of the 6 extensions of the algorithm and empirically studying their composition. Their main aim was to show that some specific combinations could give state-of the-art performances on the Atari 2600 benchmark. They have also conducted a study called as an ablation study, where they measure out the component that contributes most to the performance increase. [7] *Et al John Schulman* like our own study uses the PPO algorithm in their study, but the difference in their approach is that they have proposed a new family of policy gradient methods for RL i.e., alternating between sampling data through interaction with the environment as well as optimising a “surrogate” objective function whilst making use of the stochastic gradient descent. They test the PPO algorithm using a few benchmark tests like robotic locomotion and Atari gameplay. [8] *Et al Ziyu Wang* is using the Deuling Network Architectures; this network basically represents 2 estimators. One for used for the state value function and the other for the for the state-dependant action advantage function. They conclude by showing that their method leads to better policy evaluation, and they even managed to outperform the Atari 2600 domain. [9] *Et al Volodymyr Mnih* also developed a agent using the DQN algorithm and tested it’s efficacy on classic Atari 2600 games. They were able demonstrate that their agent was able beat the prior performance of all other agents and achieve and almost a professional human like result across a set of 49 games. We see [10] *Et al David Silver* using General Reinforcement Learning to train an agent that could master the board games known as shogi and chess. They make use of a simple *AplaZero* algorithm that can emulate superhuman performance in various domains. The algorithm was only fed basic rules of each game and was able to reach a superhuman level within just 24 hours of gameplay in chess, shogi and even *Go*. [11*] Et al Kurtland Chua* deploy their model trained with Probabilistic Dynamics Models. They did this to reduce sample complexity, However, something to note is that the performance of MBRL (Model-Based Reinforcement Learning) models is comparatively worse than model free ones i.e., MBRL models tend to learn quickly but they also tend to converge to less than optimal solutions. This paper is trying to narrow the gap between model-based and model free models. By using trajectory sampling in tandem with high-capacity neural network models incorporating uncertainty through an ensemble of bootstrapped models, they obtained a model that rivals the performance of model-free methods. In [12] *Et al Jacob Buckman* make use generalisation and regularisation in the DQN algorithm and prove that despite regularisation not being used frequently it helped the DQN learn more general features, which could then be fine tuned appropriately and reused as well. They used the Atari 2600 games as their benchmarking field. They were trying to solve the issue of RL algorithms struggling to generalise when they were evaluated within similar environments. Works [4] to [12] can also be found in table 1.0.

**3. Design and Development**

As mentioned earlier we are using unity as main development environment with supporting scripts coded in C# using Microsoft Visual Studio. We are sticking to unity version 2022.3.20f1 which isn’t latest as of writing this paper however it should adversely affect our project in any way,  
and the packages we will be using are:

ML Agents: This is the package that helps us train RL models that we will use to govern the behaviour of animals in our

wildlife simulator that we’re developing.

|  |  |  |
| --- | --- | --- |
| Reference no. | Technique Used | Performance Metrics |
| [4] | DRL, Q-learning variant | **Human performance:** [7456 pts, 31 pts, 368 pts, -3 pts, 18900, 28010 pts, 3690 pts] **Agent performance:** [4092 pts, 168 pts, 470 pts, 20 pts, 1952 pts, 1705 pts, 581 pts] |
| [5] | DRL, Double Q Learning | **Game DQN DDQN DDQN (tuned)**  Alien 7.08% 7.90% 14.50%  Amidar 7.95% 11.54% 10.29%  Assault 685.15% 564.37% 1275.74%  Asterix -0.54% 69.46% 226.18%  Asteroids -0.49% 0.98% 0.90%  Atlantis 477.77% 1884.48% 2335.46%  Bank Heist 24.82% 71.95% 138.78%  Battle Zone 47.51% 73.57% 71.87%  \* These are only a few out of the 49 games tested. |
| [6] | DRL, improved version of DQN | **Agent** **no-ops** **human starts** DQN 79% 68% DDQN (\*) 117% 110% Prioritized DDQN (\*) 140% 128% Dueling DDQN (\*) 151% 117% A3C (\*) - 116% Noisy DQN 118% 102% Distributional DQN 164% 125% Rainbow 223% 153% |
| [7] | DRL, Proximal Policy Optimisation (new policy gradient methods) | **A2C ACER PPO Tie** **(1) avg. episode reward over all of training** 1 18 30 0 **(2) avg. episode reward over last 100 episodes** 1 28 19 1 |
| [8] | DRL, Dueling DQN algorithm | **Algorithm 30 no-ops Human Starts  Mean Median Mean Median** Prior. Duel Clip 591.9% 172.1% 567.0% 115.3% Prior. Single 434.6% 123.7% 386.7% 112.9% Duel Clip 373.1% 151.5% 343.8% 117.1% Single Clip 341.2% 132.6% 302.8% 114.1% Single 307.3% 117.8% 332.9% 110.9% Nature DQN 227.9% 79.1% 219.6% 68.5% |
| [9] | DRL, DQN algorithm | Games the agent has played.  Below human intelligence: 29  Above human intelligence: 20 |
| [10] | General Reinforcement Learning, modified AlphaZero | **Game White Black Win Draw Loss** Chess AlphaZero Stockfish 25 25 0  Stockfish AlphaZero 3 47 0 Shogi AlphaZero Elmo 43 2 5  Elmo AlphaZero 47 0 3 Go AlphaZero AG0 31 – 19  AG0 AlphaZero 29 – 21 |
| [11] | DRL, Probabilistic Dynamics Models | **Problems agent comparison 1 comparison 2 comparison 4** Cart pole: 183 pts 181 pts 182 pts 182 pts 7-DOF Pusher: -25 pta -95 pts -30 pts -80 pts 7-DOF Reacher: -25 pts -90 pts -32 pts -75 pts Half-Cheetah: 7800 pts 2000 pts 6500 pts 4200 pts |
| [12] | DRL, DQN algorithm |  |

*Table 1. results of other literary works*

Apple ARKit XR Plugin: Provides native Apple ARKit integration for use with Unity's multi-platform XR API. Supports the following features: Efficient Background Rendering, Horizontal Planes, Depth Data, Anchors, Hit Testing, Face Tracking, Environment Probes, Meshing, Occlusion. The version we are using is 5.1.2.

Google ARCore XR Plugin: Provides native Google ARCore integration for use with Unity's multi-platform XR API. Supports the following features: Efficient Background Rendering, Horizontal Planes, Depth Data, Anchors, Hit Testing, Occlusion. The version we are using is 5.1.2.

AR Foundation package: A collection of Subsystems as well as MonoBehaviours and C# utilities for working with the Subsystems. Includes: Definitions of Subsystems, GameObject menu items for creating an AR setup, MonoBehaviours that control AR session lifecycle and create GameObjects from detected, real-world trackable features, Scale handling, Face tracking.

Our main idea is to build deploy a penguin in our game that can make decisions for itself such as eating fish or catching fish for its offspring. We opted to first build our RL model using ML Agents and wrote the necessary controller script for the Penguin Agent which will be learning the actions along with the appropriate considerations. A script will be written for the fish that penguin is supposed to catch to feed itself and the baby, and another for the entire area where our 3D AI assisted model will be operating. One final script will be used by our AR environment which will control the

**4. Acquisition of 3D models**

We have acquired all our 3D models from the website Immersive Limit, which has free tutorials and assets for interested learners. The models are in .fbx format which is a common way to store 3D meshes. The following 3D meshes have been imported into our project from the website:

****Penguin Agent: This is the 3D model wich will be attached to our Neural Network model trained using RL.

*Fig 1. 3D model of Penguin Agent*

A red object on a grey surface

Description automatically generatedFish: This is 3D model which will caught by our penguin agent and consumed or given to the offspring.

*Fig 2. 3D model of Fish*

A cartoon penguin on a grid

Description automatically generatedBaby Penguin: This is the 3D model which will be waiting for its parent to feed it fis

Penguin Area: This 3D model is basically a huge iceberg with small body of water in the middle where the agent will swim

around and catch fish and occasionally feed its young as well.

*Fig 4. 3D model of Penguin Area*

A white bowl with blue water

Description automatically generated

*Fig 3. 3D model of Baby Penguin*

Regurgitated Fish: This is model that will appear when the agent regurgitates fish to feed it’s young

A red and green object with arrows

Description automatically generated

*Fig 5. 3D model of Regurgitated Fish*

Heart: This 3D model will appear on the baby penguin’s head indicating that it has been fed and is full

A red heart on a grey surface

Description automatically generated

*Fig 6. 3D model of Heart*

**5. Model Training**

There are a mulititude of algorithms in the domain of reinforcement learning that we could use. Hence, we have gone with the tried and tested Proximal Policy Optimisation (PPO) algorithm. This algorithm has been dubbed as easy to use and gives good performance as well. PPO was developed by OpenAI in 2017, it is

calssified as a policy gradient method and trains an agent’s policy network. The policy network function is used by the agent to make decisions. PPO takes small step size in order for the agent to reliably reach the solution which is most optimal. Too big a step may direct the policy in an undesirable direction, too small a step lowers the efficiency of the model. To mitigate this step size issue PPO implements what is know as a clip function which constrains the policy update of any learning agent from being larger or samller than necessary. The PPO variant with this clip function is newer one with the following formula:



Where,

* Policy parameter
* Emperical expectation over time steps
* Ratio of probability under the new and old policies
* Estimated advantage of time
* hyperparamter

Using this algorithm we trained 4 different models with slightly different parameters and hyperparameters. Out of which we picked the best performing model and attached to it to out Penguin Agent. Every time we train a model we keep 8 penguin areas i.e. we have 8 penguin agents training together, all of their experiences get packed into one neural networ model, this method is also known as distributed learning. There are several benefits to this form of training, the advantages are by distributing the workload among multiple agents we can speed up the training process. Second, it is resource efficient i.e. instead of relying on one agent’s experiences we are considering the experiences of mulitple agents proving to be more cost effective and energy efficient.

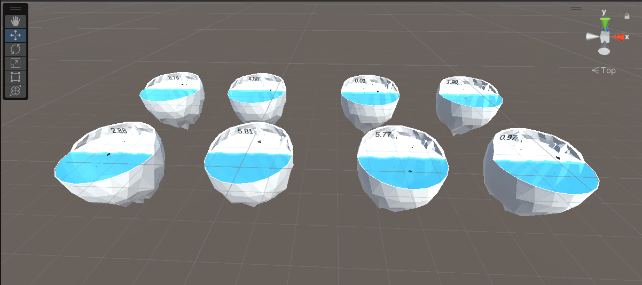
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Paramters | Model 1 | Model 2 | Model 3 | Model 4 | Model 5 |
| Batch size | 128 | 256 | 256 | 512 | 640 |
| Buffer size | 2048 | 4096 | 2048 | 4096 | 4096 |
| Learnig rate | 0.0003 | 0.0003 | 0.0003 | 0.0002 | 0.0001 |
| Beta | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| Epsilon | 0.2 | 0.1 | 0.2 | 0.1 | 0.1 |
| Lambda | 0.95 | 1 | 0.90 | 1 | 1 |
| Number of epochs | 3 | 5 | 3 | 5 | 5 |
| Hidden Layers | 2 | 3 | 2 | 4 | 5 |
| Hidden layer units | 256 | 512 | 256 | 512 | 512 |
| Maximum steps | 1000000 | 1200000 | 1000000 | 1000000 | 1500000 |
| Checkpoints | 5 | 5 | 5 | 5 | 5 |
| Time Horizon | 128 | 128 | 128 | 256 | 256 |
| Normalisation | False | True | False | True | True |

*Table 2. Parametric differences between each model*

A close-up of a computer

Description automatically generatedThe table above shows the main differences between all 5 models that were trained. The training was done the ASUS Vivobook Pro 15 with the following specifications:

Due to lack of powerfull resources, distributed learning was the method we opted for, as seen in Fig 7.



*Fig 7. Distributed learning environment*

The main bottlenecks were the CPU and RAM as they were on the cusp of thermal throttling, while the GPU itself, with 2048 CUDA cores, was sitting rather stable at about 15-20% usage during the training process.Appropriate commands were written in the command prompt of a virtual environment where the necesarry libraries such as MLAgents, Tennsorflow, Numpy were installed using pip. Tensorflow was needed because we intended to use Tensorboard in tandem while training our models, in order to chart insightful graphs that give us a cear cut comparison between all our models.We have sped up the simulation time whilst training our models, therefore our penguin agents appear to be moving at supernatural speeds, but in reality time is just flowing differently in our simulated environment within unity. Therefore, we were able to keep our maximum steps above, or at 1 million. Our goal is to determine the best model out of the five that we have trained so that we may attach the best neural network model to our penguin agent which we can then import as a prefab in our AR environment. Within this AR

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Results | Model 1 | Model 2 | Model 3 | Model 4 | Model 5 |
| Max reward | 7.766 | 7.707 | 7.733 | 7.722 | 7.75 |
| Elapsed Time | 2188.515 | 2626.218 | 2210.515 | 2260.515 | 3282.773 s |
| Policy Loss Range | 0.06413-0.07439 | 0.04378-0.05464 | 0.03873-0.0537 | 0.0281-0.03783 | 0.02256-0.03641 |
| Value Loss Range | 0.00198-0.1292 | 0.0106-0.4225 | 0.00932-0.113 | 0.00868-0.4801 | 0.02933-0.5182 |
| Entropy Range | 0.6184-1.779 | 0.9827-1.791 | 0.615-1.78 | 0.9479-1.792 | 0.6043-1.792 |

*Table 3. Results of each model*

This AR enivronment our penguin will be using this neural network model as an inference to try and catch fish.

**5. Training Results**

Following are the performance graphs we have obtained of our models using tensorboard:

A graph with a line graph

Description automatically generated with medium confidenceCumulative Reward:

8

6

4

2

0

*Loss in policy*

*Reward*

*Steps*

0 200K 400K 600K 800K 1M 1.2M 1.4M

*Fig 8. Cumulative rewards comparison*

Fig. 8 shows the comparison between all 5 models regarding rewards earned during every episode across all steps. The cumulative reward is the sum of the rewards over a series of actions or a complete episode. The maximum cumulative reward for each model is given in table 2.

A graph with colorful lines

Description automatically generatedEpisode Length:

900

700

*Length per episode*

500

*Loss in value*

300

0 400K 800K 1.2M 1.6M

*Steps*

*Fig 9. Episode Length comparison*

episode is a discrete instance of the learning process, in which an agent engages with an environment in order to accomplish a certain objective. In the graph we notice that the episode length decreases over time, this is a good sign, because it means that our agents are learning more efficiently as time goes on.

A graph of colorful lines

Description automatically generated with medium confidencePolicy Loss:

0.07

0.06

0.05

0.03

0.04

0.02

0 400K 800K 1.2M

*Steps*

*Fig 10. Policy loss comparison*

The policy loss is the expression that serves as the function for training an agent's policy network, it’s a type of NN that receives the current state of the environment as input and generates a probability distribution indicating the likelihood of each conceivable action that the agent can perform. The policy loss ranges for each model have been noted in table 2.

A graph of a graph

Description automatically generated with medium confidenceValue Loss:

0.5

0.4

0.3

0.2

0.1

0 400K 800K 1.2M 1.4M

*Steps*

*Fig 10. Value loss comparison*

The purpose of the value loss is to enhance the estimation of the value function i.e., The disparity between the anticipated value of a state by the agent's value network and the actual observed value gained from the environment is quantified by this measurement.

A graph showing different colored lines

Description automatically generatedEntropy:

1.8

1.6

1.4

1.2

1

0.8

*Entropy level*

0.6

0 400K 800K 1.2M 1.6M

*Steps*

*Fig 11. Entropy comparison*

Entropy is frequently employed as a regularisation element in the objective function during training in order to promote exploration and avoid the policy from becoming excessively deterministic. As the steps continue to run we notice that the entropy for all models is decreasing this is a good sign, as it implies that the model is exploiting its learned knowledge, putting it to use to catch the fish and exploring less.

Looking at the performance metrics of all 5 models, we decided to pick model 1. Our reasoning for this choice is that while model 1 may have the worst performance when it comes to policy loss, it beats all the models in every other performance metric. In fact its policy loss is’nt that far off either. However we did notice that model 5 was quite in comparison to model 1 based on overall performance.

**6. AR INTEGRATION**

We are now going to attaach our newly obtained neural network to our 3D mesh, a visualisation can be seen in Fig 12.

A screenshot of a computer

Description automatically generated

*Fig 12. NN model deployed.*

Now that we have deployed our model and confirmed its seamless and unbridiled function with our 3D Mesh. We can simply export the entire penguin area as a prefab along with the scripts into another project which will feature our AR enivronment.

A screenshot of a computer

Description automatically generated

*Fig 12. building AR environments in unity*

After importing we simply build our project as an android or an iOS application, here we have built an APK (Android Package Kit). And verified that the app works perfectly on multiple diffferent devices, i.e., Our penguin agent is successsfully catching fishes and feeding either itself or its offspring.

**7. CONCLUSION**

We have made use of the Proximal Policy Optimisation method and successfully trained the 3D model of a penguin to catch fish with convincing efficacy and make the decision to either feed itself or its offspring using the best model we had at our disposal. Tuning the hyperparameters seems to have introduced slight differences between each model, especially the losses. As for the cumulative reward, all the models were actually quite close to one another. The AR integration process was also seamless and was a resounding success. Therefore, we have succeeded in crafting an application acting as a wildlife simulator which uses augmented reality paired with deep reinforcement learning. With this resut we have opened a realm of endless possibilities, even our own application in its present state could become something completely different with more features and upgrades.

**8. FUTURE SCOPE**

As far as our future scope is concerned, there are plenty of new features we can add to our own application. First, we can

**REFERENCES**

***[1]*** Chengxi Li, April 2023, An AR-Assisted Deep Reinforcement Learning-Based Approach Towards Mutual-Cognitive Safe Human-Robot Interaction.   
DOI: 10.1016/j.rcim.2022.102471

***[2]*** *Robert Huy Le, June 2021, Augmented Reality and Machine Learning Incorporation Using YOLOv3 and ARKit,* DOI:10.3390/app11136006.

***[3]*** *Ghina Dandachi, May 2015, Machine learning schemes in augmented reality for features detection, DOI:10.1109/DICTAP.2015.7113179*

***[4]*** *Volodymyr Mnih, December 2013, Playing Atari with Deep Reinforcement Learning, NIPS 2013 Deep Learning Workshop,* *arXiv:1312.5602.*

***[5]*** *Hado Van Hasselt, 12 February 2016, Deep Reinforcement Learning with double Q-learning, AAAI'16: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence.*

***[6]*** *Matteo Hessel, 2018, Rainbow: Combining Improvements in Deep Reinforcement Learning,* AAAI Conference on Artificial Intelligence, 2018.

***[7]*** John Schulman,2017, Proximal Policy Optimization Algorithms, arXiv preprint arXiv:1707.06347.

***[8]*** *Ziyu Wang, 2016,* Dueling Network Architectures for Deep Reinforcement Learning, International Conference on Machine Learning.

add more animals train their NN models according to their real-world behaviour and deploy them in the game. The number of animals we could do this with is endless. Add a robust UI element that displays the information about various animals that can be seen within the AR environment. Second, we can train even more NN models and probabaly come with even better performing agents, although this will have to done for each animal separately. With above we could also release our simulator as an educator for those that would like experience the flora fauna around the world in their smartphones and upload it to the Google Play Store or Apple App Store.

***[9]*** *Volodymyr Mnih, 2015,* **Human-level control through deep reinforcement learning,** Nature.

***[10]*** *David Silver, 2018, Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm,* Science, DOI: 10.1126/science. aar6404.

***[11]*** *Kurtland Chua, 2018, Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models,* arXiv preprint arXiv:1805.12114, DOI: 10.1038/s41586-018-0024-8.

***[12]*** *Jacob Buckman,2016, Generalization and Regularization in DQN,* arXiv preprint arXiv:1611.01578, DOI: 10.1016/j.jal.2017.11.005.

**GITHUB LINK**

<https://github.com/Lazy-Dr1ft3r/PROJECT-1.git>

­