Reinforcement Learning in Unity

SUSHANTA RATNA ADHIKARI, University of Siegen, CS, [redacted]
PRAVIN PAULMONY, University of Siegen, HCI [redacted] Germany
TIMUR SEREFLIOGLU, University of Siegen, HCI, [redacted]

Reinforcement Learning is a methodology of Machine Learning which is becoming an increasing relevant in developing games. In this case we will research how to implement the RL approach using deep neural networks. In our project we use ML-Agents and teach them with a suitable supervised or unsupervised training mod. The goal is to get an ML-agent who can explore its own environment and learn for itself.[2] We will use Unity to develop ML-Agent and the Proximal Policy Optimization algorithm for training purpose.

Additional Key Words and Phrases: unity, neural networks, ml-agents, animation, reinforcement learning, PPO

1 INTRODUCTION

Initially what we are planning to do is to develop a simple learning environment in Unity with the help of ML-Agents library. The scene of the environment will be the 3D rendered model of our University and on the ground there will be randomly scattered green and red foods. The job of our agent will be to hit only the green foods. The agent will be rewarded +1 point for each green food being picked while -1 point will be deducted if the agent hits the red food. There will be various state parameters governing the agent action like velocity, direction, etc. There will be 4 discrete actions allowed to the agent which are move forward, move backward, turn right and turn left. We will implement a deep Reinforcement learning algorithm such that the neural networks will try to maximize the rewards. Later on we can add more agents to the environment and also implement different algorithms to fit such a multi-agent environment.

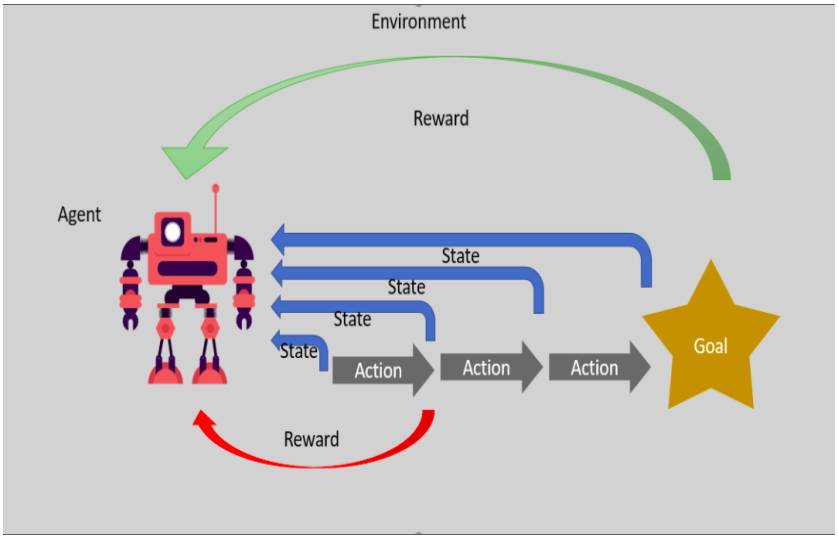

Fig. 1. Micheal Lanham. „Hands-On Deep Learning for Games.“ Apple Books.[2]

Authors' addresses: SUSHANTA RATNA ADHIKARI, University of Siegen, CS, 1532852, Siegen, Germany; PRAVIN PAULMONY, University of Siegen, HCI, 1541621, Siegen, Germany; TIMUR SEREFLIOGLU, University of Siegen, HCI, 1075805, Siegen, Germany.

The diagram shows an agent in an environment. That agent reads the state of the environment and then decides and performs an action. This action may, or may not, give a reward, and that reward could be good or bad. After each action and possible reward, the agent collects the state of the environment again. [2]

The expectations of environments in computer games are becoming more and more demanding. The environment has to meet high requirements for interactivity. A possible research question could be, how can you teach ML-agents to get to know the environment without having to explicitly program it, for example the detection of POI's in a 3D google Earth map.

2 WORKFLOW

2.1 Steps

(1) First we need to work in unity3d to create the game environment.
(2) Developing a completely new environment [4] from scratch might be challenging for us but we will try with something simpler and add complexity later on.
(3) We need to use the ML-agents package in the unity project.
(4) After writing game logic and a successful test of the game, we will build an executable [5] which will be used for training purpose.
(5) We will use a python library provided by Unity called ml-agents [6] to import the environment in a jupyter notebook.
(6) We are thinking to use simpler algorithms like Deep Q-Learning [1] and later on we may explore more algorithms like Proximal Policy Optimization (PPO).
(7) After successful training, we will test how well the agent will perform and publish the result.

2.2 Task Division

(1) *Development of Learning Agent:*
Pravin and Timur will be involved in this task. Here we have to write game logic and integrate ml-agent so that later we can train it with learning algorithm.
(2) *3D Environment via google Earth:*
Render a google Earth map, using the University Campus as game environment. Timur will be responsible for this task.
(3) *Apply Reinforcement Learning to train the agent:*
While the learning agent is being developed, Sushanta will be involved in implementation and testing of some deep reinforcement algorithms in already provided learning environment by Unity.
(4) *Integration Phase:*
Once the learning agent is developed, we all will be involved in the integration of learning algorithm to the agent for best reward.
(5) *Testing and bug fixing:*
Here again we all will collaborate for testing and further bug fixes to stabilize the overall system.
(6) *Documentation and Presentation:*
In this task too we all will collaborate for presentation of our overall work and also documenting the whole project.

2.3 Gantt Chart

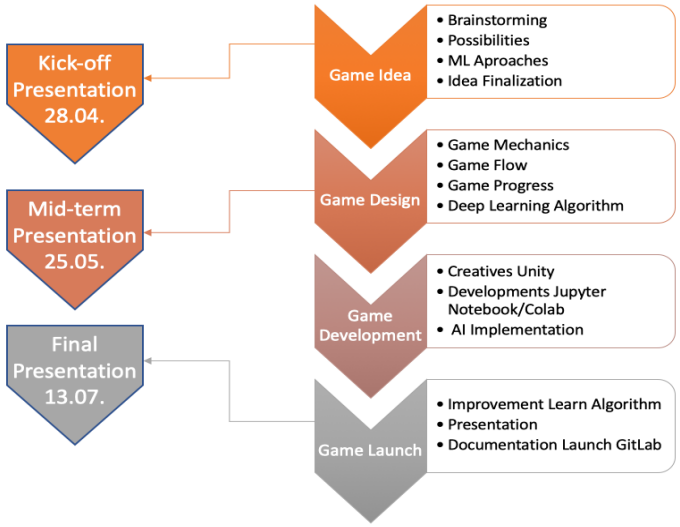

Fig. 2. Visualisation of the tasks

3 UNITY ML-AGENT

Computer games are now complex systems with their own universe. Today's games can hardly be compared with PC games from the early days. Back then, games with simple functions and loops were programmed. Computer games are becoming increasingly scientific these days. The latest trend is the use of machine learning technology. AI has long been known in the game industry, but it was rather assistant systems that are not real AI. Due to the breakthrough of machine learning, real self-learning systems are used in games today. Reinforcement learning is one of the technologies that are being used more and more.

In the modern generation 3D games play a vital role among all the people. For the development of such 3D games the unity serves as a suitable platform for both beginners and experts.

The Unity Machine Learning Agents Toolkit is an open source tool which provides the users to create games and simulations for training intelligent agents. These agents can be trained using reinforcement learning, imitation learning or other machine learning methods. These can be used as the basic approach to train the ML-Agents which serves as the basic platform of the game [7].

The Unity ML toolkit is useful for both the game developers and the AI researchers which provides multiple uses and information about the machine learning process. It will be easy to use and efficient for game development.

3.1 ML-Agent Issues

There are various ML-Agents provided by the unity toolkit package where they are attached to the unity game scene and perform some set of actions. These actions and operations can be defined from the user with the machine learning algorithms. Each and every agent has some specific characteristics and behaviour.

An agent behavior is of types such as Learning, Heuristic and Inference. The learning behaviour is like not defined but to be trained. A heuristic behavior is defined by a written set of rules which are implemented in the code. An inference behavior is the rules are defined in a file, which is typically a neural network's parameters obtained after a training process, for the agent.

While using these ml agents and also during developing our own agent for the project, we faced different kinds of issues at the beginning and during the game development process. Some of them are:

(1) At first we had some trouble with implementing the project with the agents defined due to version mismatch and also missing additional packages.
(2) The 3D background scene we rendered using google maps does not get fitted as the default background scene which created a gap between the default background.
(3) We weren't able to place the ML agents or the game objects in the proper position due to the inconsistency of the background.
(4) Also there were some issues at first while resizing and scaling the objects with respect to the background.
(5) And we faced some problems while implementing the functionality or behaviour of an agent to perform intended actions to maximize the scores.
(6) The agents did not perform the actions that they were supposed to do due to some issues in the training process which we explain more in the training section below.

By the end we came up with solutions to solve these issues and managed to define the behaviour and operations of the agents. In our case we challenged the ml-agent with two complementary tasks: hit the good foods and avoid the bad foods. Although the agent initially started with random actions, it gradually learned to avoid the bad foods and maximize the rewards by hitting the good foods.

3.2 Training ML-Agents

Training in the ML-Agents Toolkit is powered by a dedicated Python package, `mlagents`. This package provides us a command `mlagents-learn` which is the single entry point for all training workflows (e.g. reinforcement leaning, imitation learning, curriculum learning) [8]. It accepts a number of CLI options in addition to a YAML configuration file that contains all the configurations and hyperparameters to be used during training. We talk about these configurations and hyperparameters that we used to train our ml-agent in training section below. The basic command for training is:

```
mlagents-learn <trainer-config-file> –run-id=<run-identifier>
```

where

- `<trainer-config-file>` is the file path of the trainer configuration yaml.
- `<run-identifier>` is a unique name that is used to identify the results of the training runs.

In our case, for our final training run, we used the following command:

```
mlagents-learn config/trainer_config.yaml –run-id=human_20_20_6
```

The final model file (`.nn` file) is always saved under `models/` folder and we copied it to the `Assets/NNModel/` folder to use it during the inference mode.

4 METHODOLOGY

We used supervised learning with Proximal Policy Optimization (PPO), we can easily implement the cost function, do a gradient descent, and be very sure that we can achieve excellent results (figure 5) with small costs of hyper-parameter tuning. The algorithms have many moving parts that are difficult to debug, and they require a lot of optimization to get good results. With PPO, we strike a balance between simple implementation, complexity and simple optimization and calculate an update for every step. The aim is to minimize the cost function in which the deviation from previous deviations is relatively small [3].

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

4.1 Preparations

In order to ensure the functioning of the ML-Agent in Unity, some preparations are required. Aside from installing software like Unity, using the right version plays a big role. Significant problems can arise if there are conflicts with the versions of the individual components used. We ultimately use the Unity version 2019.03.15f1 with the ML-Agent Unity version unity-master. After installing all the packages, we tested various ML-Agents templates such as Food Collector and Runner. We have done many experimental attempts to find the best ML-Agent that fits our project.

4.2 Tools

Following software and libraries are used for the successful implementation of this project:

- Google Chrome (RenderDoc)
- RenderDoc
- Blender
- Unity 3D
- Python
- ML-Agents SDK (ml-agents-release_1)
- ML-Agents Python API
- Tensorflow
- TensorBoard

4.3 Developing our ML-Agent

In this section, we talk about how we developed our ML-Agent with the tool sets available to us. Initially, we explored various ml-agents which were available in the Unity ML-Agent toolkit. As this is our fresh start in the field of developing and working with ML-Agents, we decided to start with something simpler and move towards more complex ML-Agent later on. We developed a simple ML-Agent using a cube as game object and hooked it with 4 discrete actions which are:

- Move Forward
- Move Backward
- Turn Right
- Turn Left

We chose to use simple plane as our game environment during the training phase and later replace it with something more visually appealing. Then we used sphere as game object and wrote logic to spawn a number of so called good and bad foods in our game environment. The spawning logic was written in such a way that the foods are always spawned randomly in the available game space. It also handled the spawning of our agent in similar fashion. We tagged these foods as good_food and bad_food and attached those tags to the 3D ray perception sensor of the agent. This way the agent will know which type of food it is currently hitting. If the agent hits the good food, we reward it with +1 point, otherwise if it hits the bad food, we reward it with -1 point. This way the agent will learn to avoid bad foods as it starts learning in the training phase. We talk about the training step in more detail in the later section. For now we just want to mention that as the game starts, the agent action will be quite random and after several thousands iterations only it will be able to learn to hit good foods and maximize the score.

We have added an option of respawn in our game which basically handles the logic of whether the food will get respawned in a new random location after getting hit or it will just disappear

which means it got eaten by the agent. We saw that using this respawn option, the agent was able to quickly learn in a single game flow. And there are also options to choose how many good and bad foods we want in the game space.

During the training phase, the agent collects the observation vector after each action has been performed and during the inference mode, the agent uses the observation vector to perform such action that maximizes the reward. In our case the agent basically collects the measurements from the ray perception sensor and two more additional parameters which are its local velocity in x and z directions.

Once, we got confident with the workings of ml-agent, we wanted to try a bit more complex ml-agent. We chose to develop a human ml-agent and replaced our previous cube based ml-agent. The human agent has a couple of cool animations like walking, running, jumping, backwards walking, turning etc. Although, we wanted to try these actions but we decided to use continuous action space in this case and we only used forward action and turning action. These two actions take a value from 0 to 1 and based on the value the agent decides its motion. Due to the use of animations, the agent's motion is pretty smooth and appealing. Apart from changing the action space from discrete to continuous, we also change the parameters in the observation vector. Previously, we had used local velocity components in the observation vector, this time we only used the forward vector from the transform object. Later on, once the training was successful, we replaced our plane by 3D rendered University ground. In this way, we developed our ml-agent for this project.

In the following QR code, you can find our unity project. The data of the project is uploaded to a university server on gitlab (login necessary):

Fig. 3. QR - Code for the Unity project:
https://ubi22.informatik.uni-siegen.de/DataScienceAdmin/j1_unity

4.4 Using Graphics

In our project, we wanted to put an agent into a 3D model of our university. The agent was originally supposed to be able to interact with its environment, but this would require programming a lot more and preparing the 3D object of the university so that there are objects with which one can interact. We chose a simpler method and instructed an agent to collect green balls on campus. We generate a 3D model of our university campus "Unteres Schloss". The model was rendered as follows, first we used Google Chrome with a special 3d mod for the Google Maps. This mod shows all objects on the map as 3d model (not available for every City). With the software RenderDoc, we was able to capture a frame of the City. RenderDoc use this picture and generate a 3d structure for every object on this image. We then used blender to create a digital 3D model from the structure values from RenderDoc. Thus the finished 3D model of the university could be read in as an fbx file in Unity.

5 TRAINING AND RESULTS

For the training purpose, we used Proximal Policy Optimization (PPO) algorithm. This deep reinforcement learning based algorithm is implemented by Unity ML-Agent library and we can change the parameters used by the algorithm via a configuration file. In our case, most of the default parameters worked pretty well, we only had to change maximum iterations size as initially we wanted to test whether running the training process for higher iterations will make significance on the training outcome or not. Some of the key hyperparameters for the training algorithm are as follows:

- Batch Size: 1024
- Buffer Size: 10240
- Learning Rate (for gradient descent): 3.0e-4
- Number of Epochs: 3
- Number of Layers: 2
- Number of Hidden Units: 128
- Lambda: 0.95
- Beta: 5.0e-3
- Epsilon: 0.2
- Maximum steps: 6.0e6

Here, the batch size is the number of experiences in each iteration of gradient descent, while the buffer size determines the number of experiences to collect before updating the policy model. The number of layers determines the number of hidden layers we want to choose for the neural network, while the number of hidden units determines the number of units in the hidden layers of the neural network.

Other parameters are related to the PPO algorithm and their roles are now described. Lambda determines how much the agent relies on its current value estimate when calculating an updated value estimate. Beta determines the strength of the entropy regularization, which makes the policy "more random." This ensures that the agent properly explores the action space during the training phase. Epsilon influences how rapidly the policy can evolve during training.

Initially, we were not able to train our ML-Agent. We had to play around with the actions space and also with the observation vector's parameters to get better training results. We started with the greater number of good foods than bad foods to give some advantage to the agent. Although the agent was able to gain a high score, its action was pretty random. The agent was not learning to avoid bad foods but it was just hitting all foods and still it was able to get a better score as the number of good foods was high.

Seeing this behaviour, we made the number of good foods equal to bad foods and then we made multiple copies of our game environment to speed up the learning process. Even after 4 million iterations, the agent was not improving. Below is the graph (Figure 4) that shows that the agent was only able to get a score of around 20 and its behaviour was pretty random.

It took a while to figure out our mistake which was that we forgot to assign any tags to the agent's ray perception sensor. The agent was essentially blind and it was unable to distinguish between good and bad food. Once we fixed this issue, the agent quickly started learning and we had used 6 copies of our game environment and the number of good and bad foods was equal to 20. Below is the graph (Figure 5) which shows the result of our final training step. The agent was able to get the score of 150 and it was trying its best to avoid bad foods.

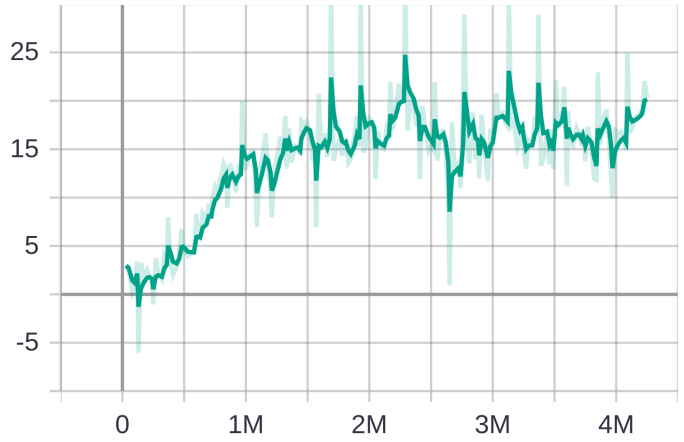

Fig. 4. Visualisation of the scores during initial unsuccessful training iterations

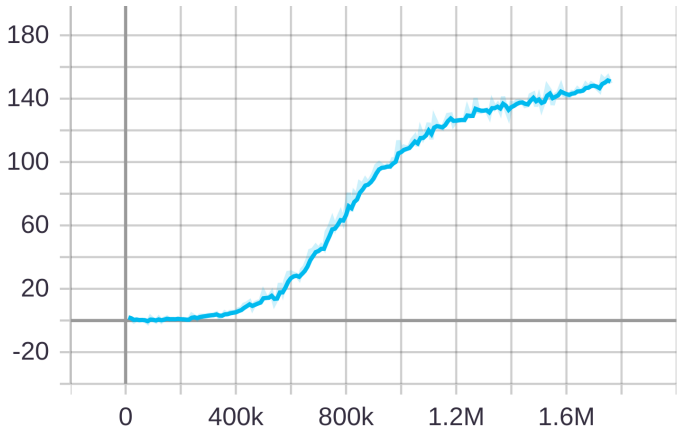

Fig. 5. Visualisation of the scores during final successful training iterations

6 CONCLUSION

Machine learning and neural networks are generally very interesting and exciting areas. These technologies can be used even better in the game industry than in real industry-related situations, because in the virtual world, there are no physical restrictions, everything can be edited and the physics can be adjusted. That is why real, industry-related simulations are also particularly good to test within a digital environment. Our work was very experimental and brought little scientific knowledge, but with the help of such simulation units, new ideas could be created and tested quite easily. This can be used as a basis for future works and should be further expanded in the scientific context. Collecting rewards was very quick to learn for our ML-Agent with PPO, comparable tests with Q-Learning were much slower. This knowledge will continue to be of particular importance to us in future work.

REFERENCES

[1] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. 2016. Continuous Deep Q-Learning with Model-based Acceleration. arXiv:cs.LG/1603.00748

[2] Micheal Lanham. 2019. *Hands-On Deep Learning for Games*. Packt Publishing, BIRMINGHAM - MUMBAI. (book).
[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347
[4] Unity. 2020. How to create New Learning Environment. https://github.com/Unity-Technologies/ml-agents/blob/latest_release/docs/Learning-Environment-Create-New.md.
[5] Unity. 2020. How to generate executable file. https://github.com/Unity-Technologies/ml-agents/blob/latest_release/docs/Learning-Environment-Executable.md.
[6] Unity. 2020. ML-Agents. https://github.com/Unity-Technologies/ml-agents.
[7] Unity. 2020. ML-Agents Overview. https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md.
[8] Unity. 2020. Training ML-Agents. https://github.com/Unity-Technologies/ml-agents/blob/release_1_docs/docs/Training-ML-Agents.md.