WAIT: Workshop on the Advances in Information Technology, Budapest, 2021

Application of Reinforcement Learning in a Multiplayer Role-playing Game

Gabor David Pasztor[1] and Marton Szemenyei[1]

[1] Department of Control Engineering and Information Technology, Budapest, Hungary

Abstract

In recent years researchers achieved breakthrough accomplishments in controlling both table and computer games with Deep Reinforcement Learning methods [15, 13]*. Although there were only a few attempts to apply these decision-making algorithms to online RPGs which take its basic concepts from real-life situations. In this paper, a complex, structured, modifiable multiplayer Role-playing Game is introduced, which provides a sufficient environment to test and train different reinforcement learning agents. The project is open-source and available on:* `https://github.com/szemenyeim/AIRPGEnv`.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Machine Learning]: Reinforcement Learning

1. Introduction

Reinforcement Learning algorithms can be used to make effective decisions or provide solutions in a complex and highly unstructured environment for everyday and industrial problems as well. However, it is highly advisable to develop and test capable algorithms in a simulated environment for security reasons. A straightforward way to simulate the environment of the agent is to use the virtual reality of computer games, since these games take most of their base concepts from the real world, so their structure is often complex enough to model real-world problems.

The main goal of the following project is to create a simple, lightweight and modifiable MMORPG (Massively Multiplayer Online Role-playing Game) Game with basic graphic design and functionality, and train Deep Reinforcement Learning Algorithms on it.

The main requirements are the following:

- Hero shall be controllable in the game environment.
- Multi-client architecture.
- Simple Artificial intelligence for Non-Playable Characters (NPCs).
- Basic development.
- Quest options with NPCs.
- Modifiable, multi-platform environment.
- Easy to train.
- Open-source

The purpose of the project is not only to develop an environment but to challenge multi-agent reinforcement learning algorithms in it. The experiences about the first models can be found in Section 5.

Our main contributions are the following:

- Creating an environment that is massively multiplayer and can be used from multiple training machines
- Train and compare popular Deep Learning algorithms
- Show that environment can be learned to some degree

2. Related Work

2.1. Reinforcement Learning

When thinking about the nature of learning, the idea of learning by interacting with the environment is one of the first thoughts to occur. *Reinforcement Learning* is a computational approach to learning from interactions, by creating an action-state map to maximize the numerical reward signal and find an applicable policy.

One of the challenges in Reinforcement Learning is the trade-off between exploration and exploitation. To maximize the reward, the reinforcement learning agent should prefer actions that were effective before, but it has to try a vast number of different actions to discover good ones. Therefore, the agent needs to exploit its experiences, while it has to explore new opportunities to find better actions for the future. The dilemma is neither strategy can be pursued exclusively without failing the task. This dilemma has provided many challenges for researchers for many decades, yet remained unresolved [1].

2.2. Elements of Reinforcement Learning

Four main elements of the reinforcement learning system can be identified:

1. A **policy** is -roughly speaking- a mapping between the states of the environment and the actions to take. The policy is the core of the agent as it defines the behaviour of the learning agent at the given time. In general, policies may be stochastic.
2. **Reward signal** describes the immediate, intrinsic desirability of the environmental state.
3. On the other hand, the **value function** is used to specify what is desirable in the long run. It describes the total amount of reward an agent can expect to accumulate over the future, starting from the given state. It is a more far-sighted and refined judgment of how pleased the agent is with that particular state of the environment. Actions are chosen based on the value judgment instead of the immediate reward. Although it is much harder to determine the value function, as it will be estimated according to the sequence of the observations the agent makes over its entire lifetime. One of the most important components of all reinforcement learning algorithms to find an efficient method to estimate the value function.
4. With the help of the **model of the environment**, the agents can plan the possible reactions for future states before they occur. In the realm of reinforcement learning algorithms, there are model-based methods, which use models and planning, and model-free methods, which are explicitly trial-and-error learners.

2.3. Deep Reinforcement Learning methods

In recent years deep reinforcement learning algorithms used to surpass human-level performance in many computer games [8, 9]. These Atari games have a small action space, therefore it can be effectively controlled by value-based algorithms such as Deep Q-learning, which is an extension for traditional Q-learning [16] in order to estimate the future state-value tuples.

However, the performance of these value-based decreases significantly by high-dimensional action spaces, thus *Policy Gradient Methods* are used in these environments, which learns both a policy and a state-value function. A subclass of these methods is often called *actor-critic methods*, where the learned policy is referred by 'actor' and the value function by 'critic'. Better results can be achieved by parallel execution of the learning agents, so the can experience different states in the given timestamp [7]. One of the state-of-the-art approaches is Proximal Policy optimization [12], which performs even better, by implementing Trust Region update in a simple and adaptive way, creating a good balance on the speed of optimization, performance and implementation.

Deep reinforcement learning has been applied in various games [4] e.g.: in 3D First Person Shooters [6, 3] or text-based games [10]. Still, the most cutting-edge breakthrough is that DeepMind's deep learning agent achieved to reach grandmaster level in Starcraft II, a game with one of the most complex mechanics [15].

Furthermore, computer games provide plenty of useful information for deep learning research as they are huge playgrounds based on the real world with an infinite amount of modification, provided by the customizable game environment [11].

OpenAI Gym also provides an MMORPG (Massively Multiplayer Online Role-playing Game) environment for the massive amount of online agents. However, this environment provides an arena for players to fight and compete against each other, but without Quests, NPCs or sophisticated tasks the agents won't cooperate to complete a goal, which is one of the main focus of our project. [14]

3. General Framework

Before the specific implementation is introduced, it is important to give a perspective on the key ideas and functionalities this game was built on. The main purpose of this game environment is the provide a customizable environment for deep learning agents where they can act in different roles based on their reward function. This approach will provide a tool to see how agents using different policies can perform compared with each other in a shared environment. The agents have finite resources in a shared map, where the main goal is to eliminate every hostile character. The enemies have numerical superiority, and therefor the agents can not defeat them simply, they need to develop an advanced strategy.

The environment should provide different strategies for the agents to complete the game: they can defeat the opponents together as allies, or they can fight against each other till they collect enough experience points to overpower every other character. The framework also provides friendly characters who could help the agents and give them more power, so they could go on there own way without the need to rely on other agents. The implementation should be balanced on a way that neither behaviour described above should be supported against the other. Another idea behind the game logic is that the map could affect the behaviour of the players with forbidden regions. Therefore the maps need to be easy to create, add or modify according to the user's needs.

4. Implementation

In this chapter, the environment is introduced with the main components and functionalities in particular details. The whole environment is implemented with Python3 and consists of two main parts: the Server and the Client.

4.1. Client

4.1.0.1. OpenAI Gym The proposed environment is compatible with OpenAI Gym [2], which is a popular toolkit that facilitates the development of Reinforcement Learning environments. Gym provides a standardized interface for training agents and also has a large number of built-in environments.

The main task of the client is to provide a Graphical User Interface (GUI) and manage the rewards. The GUI is based on OpenCV, as it has basic drawing capabilities, suitable for creating semantic maps.

Every player can see a 64×64 pixel territory from the current map (Fig. 2) and a 16×16 pixel mini-map (Fig. 1). These two images make up the observation space for the Agents. The mini-map is blank at the beginning when a player spawns. The player proceeds by exploring the available territories, after which the down-scaled map can be shown. To help the exploration process, experience point rewards are given for every newly discovered pixel.

The action space is a 6-dimensional discrete variable: The player can move in the world with keys (W, A, S, D) in four directions. To manipulate the surrounding elements, the player can attack (SPACE) or interact (F).

Experience rewards are provided for exploration and successfully defeating creatures. The client calculates the reward for the agent in every step based on the experience points earned and the health points lost as follows: if the agent dies, the reward is -1. If the agent gains XP, the reward will vary between 1 and 3 according to based on the reason for the XP gain. Exploration gives 1, hitting a standard enemy gives 2 while completing a quest gives 3 Experience Points. The maximum health is 100 for every player at every level. If the player loses all health points, they die and re-spawn automatically in a random position.

4.1.0.2. The Game Elements:

- **Blue Territory** is water. Players are not allowed to go in the water as they cannot swim. On the other hand, monsters can move through the water.
- **Red Dots** represent players on the map. The player's agent is usually in the middle of the window.
- **Black Dots** are the Non-Playable Characters. NPCs are stationary and they are generally not hostile, allowing players to interact with them. As a result of the interaction, the players can get quests. Currently, these quests are monster hunts, and the targets are marked on the mini-map with a red rectangle.
- **Blue Dots** are the monsters. They will engage and attack the players nearby.
- **Purple Dots** on the main map are the marked enemies from the quests. If the player kills them, it will provide extra experience.

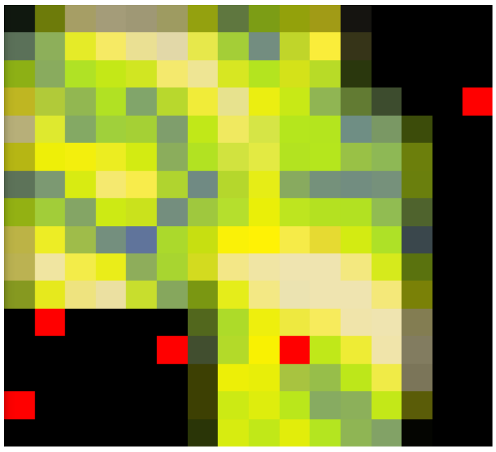

Figure 1: *Mini-map showing the blurred game environment and the position of the marked enemies as red squares.*

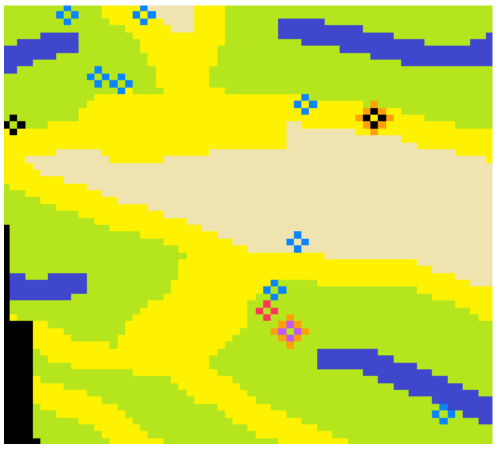

Figure 2: *The territory of a single Player. The red dot represents the hero, whereas the black dots are the NPCs that can give hunt-quests. The blue dots are the standard enemies. The orange glare around the black dot means that the hero interacted with it, and it gave a quest to the hero and the special, targeted enemies are marked with purple.*

4.2. Server

The server is responsible for managing and supervising the entire game mechanism by calculating every player interaction based on the pressed keys. Furthermore, it is also tasked with handling combat, experience points, quests and NPCs as well. The server is organized into 3 threads:

- **Main Thread:** This thread handles the game logic, processes the messages, and executes the commands from the players.
- **Server Thread:** It handles communication with the clients. Whenever a new message is received, it is placed in a queue for further processing by the main thread.
- **Monster Thread:** This thread is devoted to processing enemy AI: the NPCs that can harm the Players. If a player approaches an enemy, it will follow the player and try to hit it if it is near enough.

The Server can display the whole game map for debugging purposes during the training, so agents can be observed. Three different environments are used for the initial tests. An *arctic* with relatively few inaccessible areas, a *lake* with large bodies of water and the *valley* with a labyrinth-like landscape as it can be seen in Fig. 3. The first stage contains 50 Monsters and 10 NPCs.

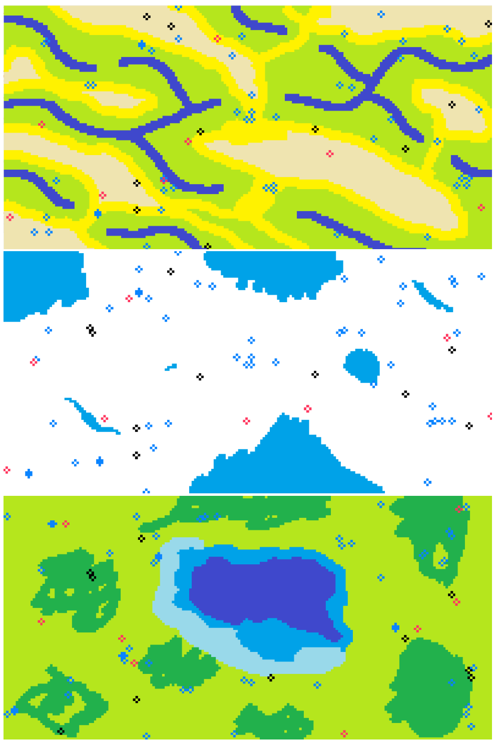
Figure 3: *The 3 different maps of the game with 8 players.*

5. Training

After the first successful test runs, the environment was tested with multiple reinforcement learning agents. For the training, the Advantage Actor-Critic (A2C) [5] and Proximal Policy Optimization (PPO) [12] algorithms were used.

For every test, certain requirements need to be made to know whether the test is successful. In this initial training, the agents need to explore the game territory, without getting stuck in a known location. Moreover, the actors should protect themselves from the monsters when attacked. In addition, we hoped that the agents would fight with each other or co-operate to eliminate multiple monsters. As our last requirement, the agents should learn to pick up and accomplish quests for higher rewards.

5.1. Performance

The running loss is calculated as follows:

$$L = 1.0 * L_{act} + 0.5 * L_{crit} - 0.001 * L_{entr} \quad (1)$$

Where L_{act} and L_{crit} are the actor and critic losses respectively, and L_{entr} is the entropy-based regularization term. With the help of a multiprocessing vector environment, 8 agents were trained in parallel. The agents have 50 enemies around the map, with 5 NPCs to get quests from. After they eliminated every monster, all the 50 monsters respawn again with a higher level and in a different map. Thus the fewer the enemies are, the more exploration needed.

The neural network uses features from both the surroundings of the agent and the minimap. The minimap helps them to find new opponents and explore new territories. Without that, they tend to "vibrate" in one place, without any progress. The used network architecture can be seen in Fig. 4.

The initial test was performed with advantage actor-critic method, and it provided some useful experiences. Although there wasn't any sufficient exploration policy, the learning rate accelerated in time as 5 shows. The reason for this is, that higher level monsters provide more experience points, however, the stronger monsters cannot be defeated so easily. The plateaus in the learning curves mean the lack of enemies. When most of the monsters are hunted down, the ones left behind cannot be found easily by the learning agents, so they will learn only from the exploration reward. 5 shows that the plateaus shorten over time, as the agents tend to fight against each other because they will not compete with the overpowered and rare enemies.

To solve the problem of overpowered enemies balance was needed between the strength of the characters. Therefore, the attack damages are modified to be the same for the heroes and monsters as well during the whole game. However, the enemies were still overpowered because of the numerical superiority they had.

The test training was performed both with A2C and PPO algorithms because one of the main goals of this project is to provide an environment to create, test and compare modern reinforcement learning algorithm. According to the figures, the performance of the PPO algorithms is better as PPO-agents are able to collect more reward and learn strategies

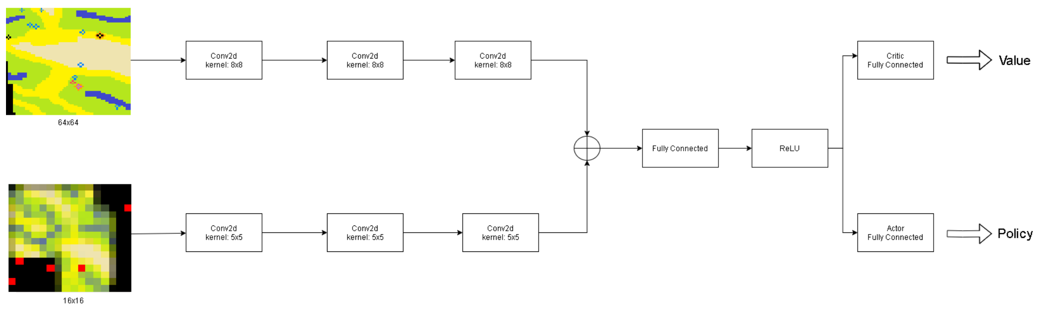

Figure 4: *Deep Neural Network architecture used for the tests.*

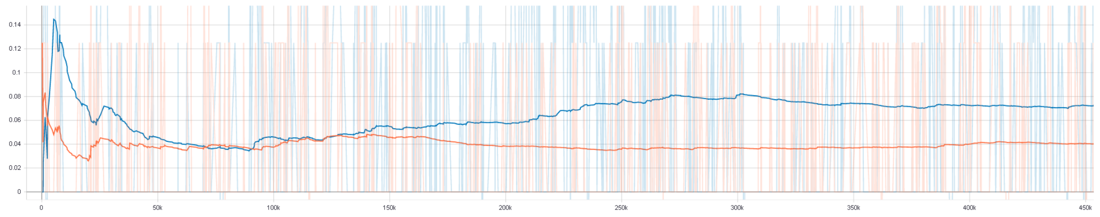

Figure 5: *Mean Reward: PPO - blue, A2C - red*

earlier. This is in alignment with the different tests published in [12], which claims that proximal policy optimization has better overall performance, especially in more general settings.

According to the log files, the A2C-agents completed 128 levels, whereas PPO-agents completed only 66 levels during 450000 episodes, despite that PPO-agents collected more reward on average. This is because A2C agents explored more territory in the game and therefore they could get more experience points before fights and evolve more, whereas PPO-agents were more engaged in combat without the proper level, and fearless attacking of enemies is more rewarding.

6. Experimental Results

The detailed description of the test results is described in the following section. To get a complete picture of the performance of the environment, it is crucial to see how much reward has been collected by the agents during the different runs. The data in Table 1 shows the mean values from 5 different runs, each with 500,000 episodes and with different random seed values. The columns of Table 1 are the following:

- ***Level:*** The experience level of the agent on the last map before the game ended.
- ***XP:*** The experience points the agent collected on the different levels.
- ***Re-spawn:*** The number of re-spawns. It means how many times the agent was defeated by the monsters or other heroes during the game.
- ***Kill:*** The agent's kill count.
- ***Quest:*** The number of quest-related kills.

The log shows that the A2C agents explored more territory in the game and therefore they could get more experience points and less defeat before fights and evolve more, whereas PPO-agents were more engaged in combat thus they could kill more enemies and complete more quest hunt, as it is mentioned before, at the end of Section 5.1.

With random policy, the agent could collect 168,226 XP and neutralize 78 monsters where 8 monsters were killed as a Quest, and the random agent will fall around 31 times in 500000 episodes.

It is worth to take a look at the *Kill-Quest* ratio. The agents took only a few quests with either of the algorithms, although the quests provide more reward. Human players can usually complete a map with 50:35 *Kill-Quest* ratio. This means that the reward system needs to be redesigned to encourage more specific Quests rather than the standard kills or the exploration. The exploration should be only a tool for finding the enemies in complex maps.

7. Conclusion

In this paper, we introduced a Role-playing game environment, which allows researchers to train and test deep reinforcement learning algorithms with multiple agents at the same time. The agents can learn different strategies based

Table 1: *Mean values of data of the runs per agent*

Agent	PPO				
	Level	*XP*	*Re-spawn*	*Kill*	*Quest*
1	450	10.15M	2,518	2,742	8
2	460	10.53M	2,489	2,722	6
3	457	10.44M	2,540	2,676	8
4	456	10.32M	2,557	2,696	6
5	451	10.22M	2,527	2,744	10
6	455	10.24M	2,493	2,650	8
7	468	10.90M	2,603	2,836	11
8	468	11.00M	2,463	2,747	7
Agent	**A2C**				
	Level	*XP*	*Re-spawn*	*Kill*	*Quest*
1	369	6.63M	896	1,482	48
2	377	7.18M	916	1,475	38
3	375	6.88M	889	1,484	46
4	377	7.06M	850	1,464	44
5	376	7.27M	880	1,471	47
6	373	6.98M	894	1,473	36
7	377	7.18M	848	1,500	51
8	363	6.60M	876	1,496	44

on reward settings in a structured and partially observable environment. After the introduction of the game, we also performed tests with different reinforcement learning algorithms.

The established environment functions as a sandbox environment for multiple agents and provides a Role Playing Game-like world for them. After the first successful test, the next goal is to train the agents in more complex structured maps, with more activity and a larger amount of characters. A more sophisticated development policy will also be available for the characters with a considerably more complex reward system. The agents also will be able to use different tools and abilities. The project is available in the following link: `https://github.com/szemenyeim/AIRPGEnv`

References

1. Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. *Machine Learning*, 47(2/3):235–256, 2002.
2. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.
3. Jack Harmer, Linus Gisslén, Jorge del Val, Henrik Holst, Joakim Bergdahl, Tom Olsson, Kristoffer Sjöö, and Magnus Nordin. Imitation learning with concurrent actions in 3d games, 2018.
4. N. Justesen, P. Bontrager, J. Togelius, and S. Risi. Deep learning for video game playing. *IEEE Transactions on Games*, 12(1):1–20, 2020.
5. Vijay R. Konda and John N. Tsitsiklis. Onactor-critic algorithms. *SIAM J. Control and Optimization*, 42:1143–1166, 2003.
6. Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. *CoRR*, abs/1609.05521, 2016.
7. Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.
8. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.
9. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.
10. Karthik Narasimhan, Tejas D. Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *CoRR*, abs/1506.08941, 2015.
11. Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.
12. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.
13. David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, and Adrian et al. Bolton. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.
14. Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural MMO: A massively multiagent game environment for training and evaluating intelligent agents. *CoRR*, abs/1903.00784, 2019.
15. Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, and Petko et al.0 Georgiev. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
16. Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, May 1992.