FlashRL: A Reinforcement Learning Platform for Flash Games

Per-Arne Andersen Morten Goodwin
Ole-Christoffer Granmo
University of Agder, Faculty of Engineering and Science
Serviceboks 509, NO-4898 Grimstad, Norway

Abstract

Reinforcement Learning (RL) is a research area that has blossomed tremendously in recent years and has shown remarkable potential in among others successfully playing computer games. However, there only exists a few game platforms that provide diversity in tasks and state-space needed to advance RL algorithms. The existing platforms offer RL access to Atari- and a few web-based games, but no platform fully expose access to Flash games. This is unfortunate because applying RL to Flash games have potential to push the research of RL algorithms.

This paper introduces the Flash Reinforcement Learning platform (FlashRL) which attempts to fill this gap by providing an environment for thousands of Flash games on a novel platform for Flash automation. It opens up easy experimentation with RL algorithms for Flash games, which has previously been challenging. The platform shows excellent performance with as little as 5% CPU utilization on consumer hardware. It shows promising results for novel reinforcement learning algorithms.

This paper was presented at the NIK-2017 conference; see http://www.nik.no/.

1 Introduction

There are several challenges related to developing algorithms that can interact with human-level performance in real-world environments, such as computer games. Researchers often use toy experiments when working with *Reinforcement Learning* (RL), because it is easier, cheaper and consumes less time to orchestrate. With several applications for RL in daily life, it has become an essential field of research [13, 4]. However, existing learning platforms for games have major limitations such as few game environments and little environment control.

OpenAI is a non-profit company that is currently one of the leading researchers of RL. OpenAI *Universe* is a software platform that has several game environments aimed at artificial research. The problem with this software is that individual developers are not directly permitted to supplement new environments to the repository, and there is little documentation on how to contribute to new environments. *FlashRL* changes this with our proposed architecture as the control is given back to each researcher.

Adobe Flash is a multimedia software platform used for the production of applications and animation. The Flash run-time was recently declared deprecated by Adobe, and by 2020, no longer supported. Flash is still frequently used in web applications, and there are several thousand games created for this platform. Several browsers have removed support for Flash, making it impossible to access the mentioned game environments. Games have proven to be an excellent area of machine learning benchmarking, due to size and diversity of its state-space. It is therefore essential to preserve Flash as an environment for reinforcement learning.

Automating Flash applications is a relatively untouched area. The technology has been succeeded by several better options for web development, for example, HTML5. This makes it hard for algorithms to control Flash environments programmatically. There are already reinforcement learning platforms that support Flash games as part of their game library, but these use browsers to execute the Flash run-time.

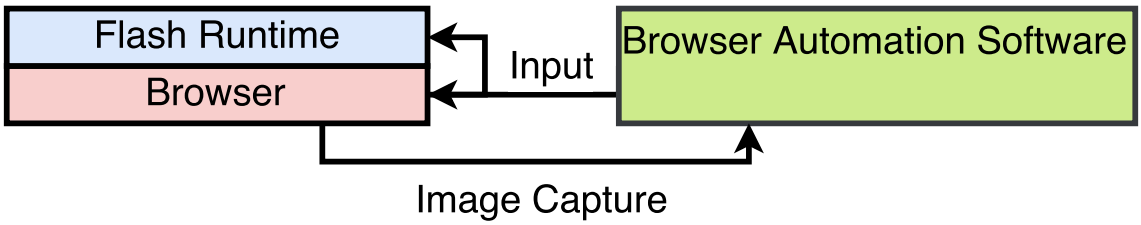

Figure 1: Interacting with Flash through browser automating

Figure 1 illustrates how interaction with the Flash environment would typically be carried out through browser automation software such as *Selenium*. *Selenium* can automate most modern browsers. It does not directly support Flash automation, but can easily be used for this purpose with minimal customisation [3]. With the loss of browser support, the difficulty of controlling Flash applications increases, and there is a significant risk that excellent game environments for reinforcement learning are lost.

FlashRL is unique for reinforcement learning as it allows researchers to use any desired Flash environment. It gives full control of the game environment and is not based on running Flash applications in the browser.

FlashRL is targeted research in reinforcement learning, but can also be used in other machine learning algorithms. It supports all kinds of Flash applications but is primarily used for agent-based gameplay. Several thousand game environments are included in the first release of the software[1]. Multitask 2 is a Flash game that is excellent for reinforcement learning as it requires the agent to perform several tasks simultaneously. We show in this paper that our learning platform can be used to train novel reinforcement algorithms without any customisation.

In Section 2, we discuss related work for existing learning platforms in machine learning. We also argue why web browsers are no longer viable as Flash runtime. Section 3 briefly outline what reinforcement learning is and explains how Q-Learning works. Section 4 outlines the proposed platform and thoroughly describe its underlying architecture. In Section 5 we show initial results of utilizing the proposed learning platform for reinforcement learning. At Section 6 summarises the work and argue why the proposed learning platform is used for reinforcement learning research. Section 7 outlines a road-map for further development of the platform.

2 Related Work

With the increasing popularity in RL, there is a need for flexible learning platforms. Several learning platforms exist that can run a limited number of games, but no platform that features an open-source interface with possibility to run *any* Flash game.

Bellemare et al. provided in 2012 a learning platform *Arcade Learning Environment* (ALE) that enabled scientists to conduct edge research in general deep learning [1]. The package provided hundreds of Atari 2600 environments that in 2013 allowed Minh et al. to do a breakthrough with Deep Q-Learning and A3C. The platform has been a key component in several breakthroughs in RL research. [11, 9, 8]

In 2016, Brockman et al. from OpenAI released GYM which they referred to as *"a toolkit for developing and comparing reinforcement learning algorithms"* [2]. GYM provides various types of environments from following technologies [2]: Algorithmic tasks, Atari 2600, Board games, Box2d physics engine, MuJoCo physics engine, and Text-based environments. OpenAI also hosts a website where researchers can submit their performance for comparison between algorithms. GYM is open-source and encourages researchers to add support for their environments.

OpenAI recently released a new learning platform called *Universe*. This environment further adds support for environments running inside VNC. It also supports running Flash games and browser applications. However, despite OpenAI's open-source policy, they do not allow researchers to add new environments to the repository. This limits the possibilities of running any environment. Universe is, however, a significant learning platform as it also has support for desktop games like Grand Theft Auto IV, that allow for research in autonomous driving [7].

Selenium is a software for automating web browsers and is used primarily for unit-testing of web content. There were some efforts to create a version that allowed to interact with Flash content, but it was quickly abandoned. There is limited support for interacting with Flash, by selecting the DOM-Element in HTML and sending

[1]Author of this paper takes no credit for any game environments

key-presses via Javascript. Several learning platforms utilize this method, but due to the deprecation of Flash in browsers, it is no longer a viable option.

3 Reinforcement Learning

Reinforcement learning can be considered hybrid between supervised and unsupervised learning. We implement what we call an agent that acts in our environment. This agent is placed in the unknown environment where it tries to maximize the environmental reward [14].

Markov Decision Process (MDP) is a mathematical method of modeling decision-making within an environment. We often use this technique when utilizing model-based RL algorithms. In *Q-Learning*, we do not try to model the MDP. Instead, we try to learn the optimal policy by estimating the action-value function $Q^*(s, a)$, yielding maximum expected reward in state s executing action a. The optimal policy can then be found by

$$\pi(s) = argmax_a Q^*(s, a) \tag{1}$$

This is derived from *Bellman's Equation*, because we can consider $U(s) = max_a Q(s, a)$, the utility function to be true. This gives us the ability to derive following update-rule equation from Bellman's work:

$$Q(s,a) \leftarrow Q(s,a) + \underbrace{\alpha}_{LearningRate}\left(\underbrace{R(s)}_{Reward} + \underbrace{\gamma}_{Discount factor}\underbrace{max_{a'}Q(s^{'},a^{'})}_{NewEstimate} - \underbrace{Q(s,a)}_{OldEstimate}\right) \tag{2}$$

This is an iterative process of propagating back the estimated Q-value for each discrete time-step in the environment. It is guaranteed to converge towards the optimal action-value function, $Q_i \rightarrow Q^*$ as i $\rightarrow \infty$ [14, 10]. At the most basic level, Q-Learning utilize a table for storing $(s, a, r, s^{'})$ pairs. But we can instead use a non-linear function approximation in order to approximate $Q(s, a; \theta)$. θ describes tunable parameters for approximator. Artificial Neural Networks (ANN) are a popular function approximator, but training using ANN is relatively unstable.

4 Flash Reinforcement Learning (FlashRL)

The proposed platform is an interface that acts as a bridge between the *Gnash Flash player* and the reinforcement learning algorithms. *Flash Reinforcement Learning* (FlashRL) is a new platform that allows researchers to run algorithms on any Flash-based game efficiently.

The learning platform is developed primarily for the operating system Linux but is likely to run on Cygwin with few modifications. There are several key components that FlashRL uses to operate adequate, see Figure 2. It uses a Linux library called XVFB to create a virtual frame-buffer that is used for graphics rendering [6]. Inside this frame-buffer, a Flash game chosen by the researcher is executed by a third party flash player, for example, *Gnash*. A VNC server serves the XVFB frame-buffer and allows FlashRL to access it by utilizing a VNC Client. The VNC Client can then issue commands like keyboard presses and mouse movements. The VNC Client *pyVLC* was specially made for this learning platform. The code base originates from python-vnc-viewer [15]. The last component of FlashRL is the Reinforcement

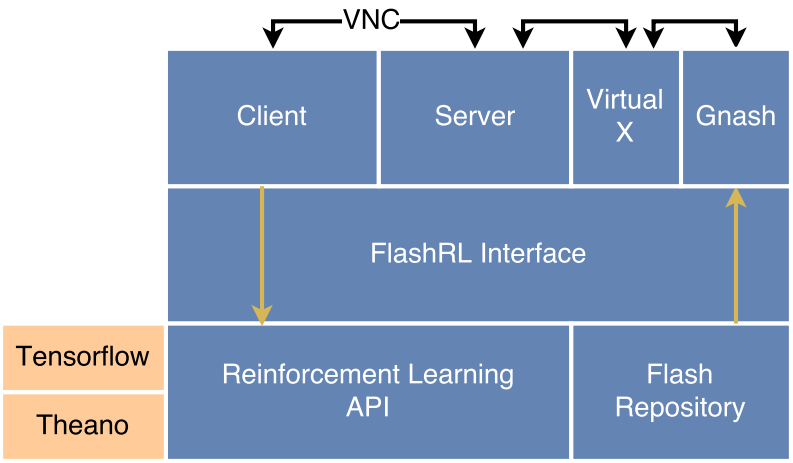

Figure 2: FlashRL Architecture Overview

Learning API that allows the developer to access the input/output of the VNC client. This makes it easy to develop sequenced algorithms by using the API callbacks or manually by threading.

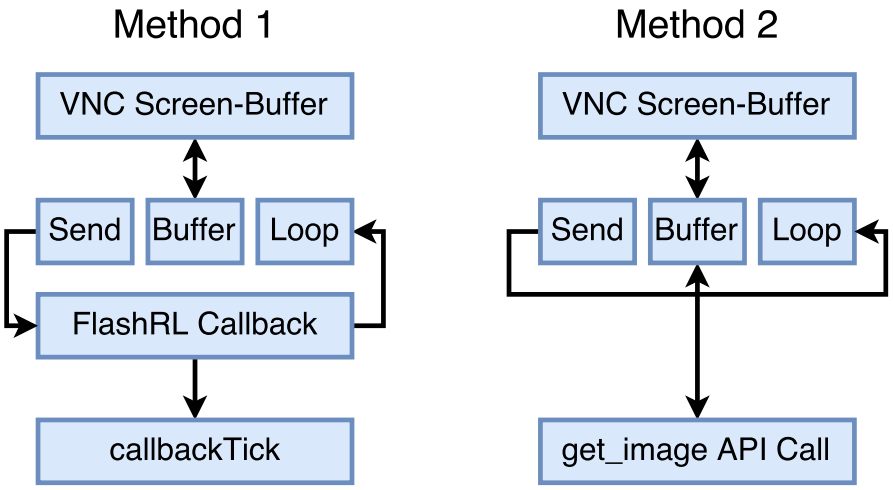

Figure 3: Frame-buffer Access Methods

Figure 3 illustrates two methods of accessing the frame-buffer from the Flash Game. Both approaches are sufficient to perform reinforcement learning, but each has its strength and weaknesses. Method 1, seen in Figure 3 allows the developer to get frames served at a fixed rate, for example, 60 frames per second. Method 2 does not restrict the frequency of how fast the frame-buffer is captured. This is preferable for developers that do not require images from fixed time-steps as it requires less processing power per frame. The framework was developed with deep learning in mind and is proven to work with Keras and Tensorflow.

Several thousand game environments are shipped with the initial version of FlashRL. These game environments were gathered from different sources on the web. FlashRL has a relatively small code-base and to preserve this size, all of the Flash games are hosted remotely. The quality varies, and some of the games are not tested or labeled. Most games are however tested and can be played without issues,

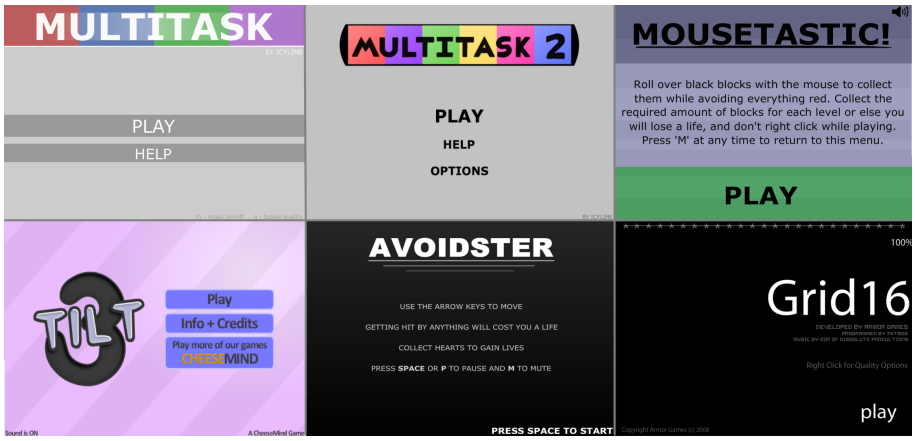

Figure 4: Selected environments from the FlashRL game repository

see Figure 4.

5 Experiments

This section presents experiments of reinforcement learning algorithms applied in FlashRL. We use the game Multitask 2 [2] to test the learning platform. Multitask 2 was chosen because it challenges the algorithm to master four different mini-games simultaneously.

The experiments are grouped in two. The first experiment determines the hardware requirements of the platform and benchmarks the speed of critical operations. The second experiment is an implementation of standard Deep Q-Learning trained on raw state images from Multitask 2 to perform game actions. The latter is meant as a proof of concept that RL algorithms can be applied in FlashRL.

All experiments were conducted on Ubuntu Linux 17.04 x64 running Python 3.5.3. The machine has 64GB memory, Nvidia GeForce 1080TI, and Intel I7-7770k as hardware.

Multitask 2

Figure 5 illustrates the game-play of Multitask 2. The game is split into four-game phases. The first phase (lower right corner in Figure 5) is a single paddle that the player must balance a ball on. In state two (lower left corner in Figure 5) , the player must control the second paddle to avoid arrows traveling towards it. The third phase (upper right corner in Figure 5) consist of an arrow with mechanics relatable to the game Flappy Bird [12]. In the final phase (upper left corner in Figure 5), the player must additionally jump over holes on the ground. For the player to succeed the game, he must control eight actions simultaneously. The score is calculated by adding a single point for each second survived in the game.

Experiment 1: Hardware Requirements

Recall from section 4 that there are two methods of accessing the frame-buffer. The first method (Method 1) is based on retrieving the frame-buffer at fixed time

[2] Multitask 2 - http://multitaskgames.com/multitask-2.html

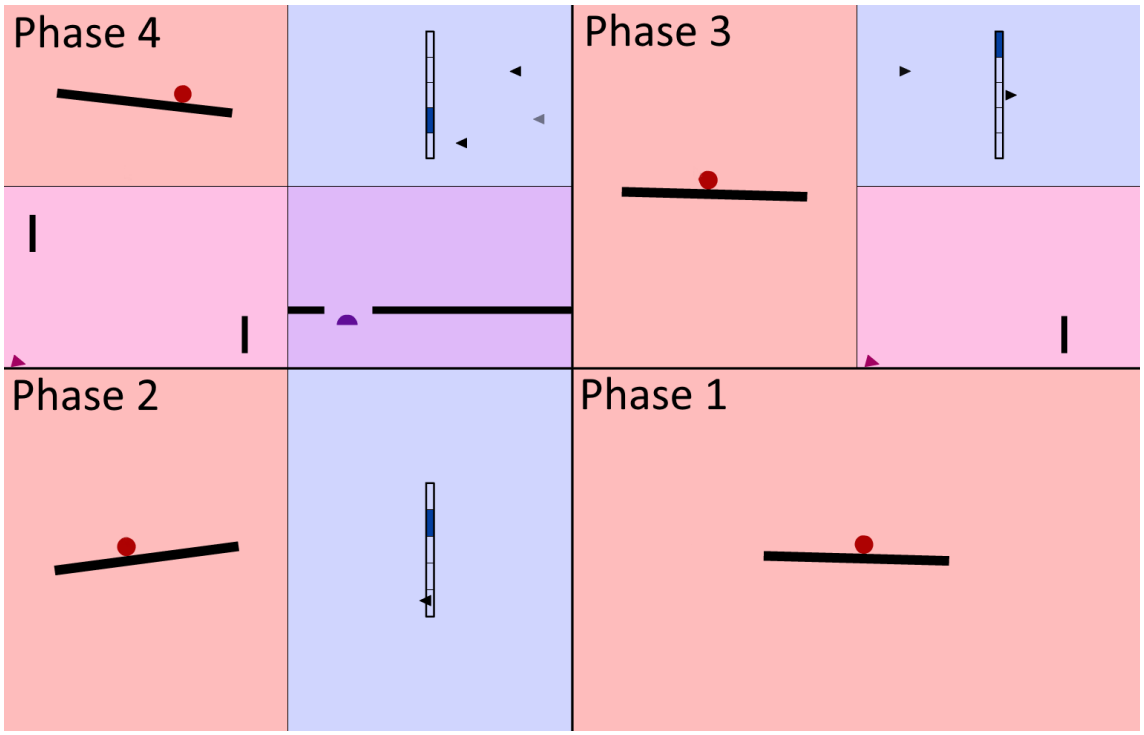

Figure 5: In-game footage of the game Multitask

intervals. The second method (Method 2) does not have any interval restriction. This makes Method 2 faster because it does not require sleep between frames. This causes the framework to consume all available CPU, which is not always preferable.

We can see from Figure 6 that using Method 1 with the interval set to 30 fps uses approximately 5% of the CPU. Increasing the interval to 300 increases it to 13%. We gradually increased the interval until the CPU ran at maximum. A single I7-7700k can compute approximately 6300 fps images from the frame-buffer before struggling to keep up.

The GPU Did not recognize any load during these test because the Flash environment is software rendered. Memory consumed were between 200MB and 500MB depending on the speed. We believe that the reason for memory increase is that Python does not garbage collect old frame-buffer snapshots between iterations, and therefore gets an increased memory load.

Experiment 2: Reinforcement Learning

Deep Q-Network (DQN) is a novel algorithm architecture developed by Minh et al. at Google DeepMind. It combines Q-Learning estimating Q-Values from a neural network. [11]

In our tests we used Double Q-Learning from Hasselt et al. [5]. We also used Dueling from Wang et al. that increases the learning precision by using two estimators: state-value and action-advantage function [16]. We used a discount factor of 0.99, learning rate of 0.001 and mini-batch of 16. We used exploration/exploitation strategy with ϵ-greedy where it started at 0.9 and finished at 0.1. The ϵ annealing was set to 10 000 steps. This is a relatively low epsilon phase. But it seemed to work well in this environment.

Figure 7 illustrates the training of DQN, where the x-axis represents episodes

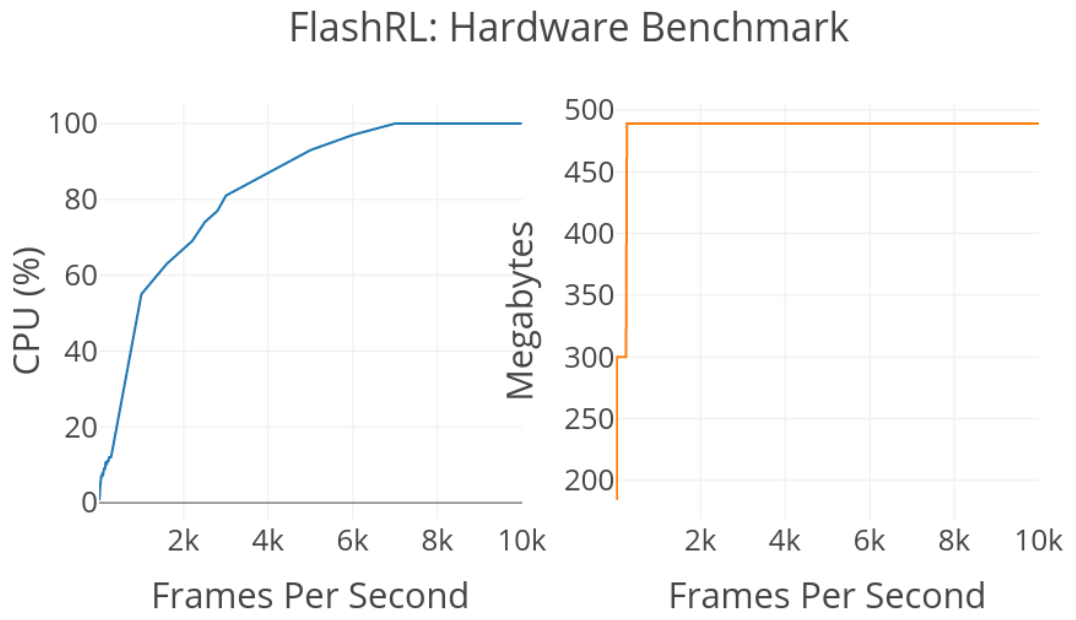

Figure 6: Hardware benchmark

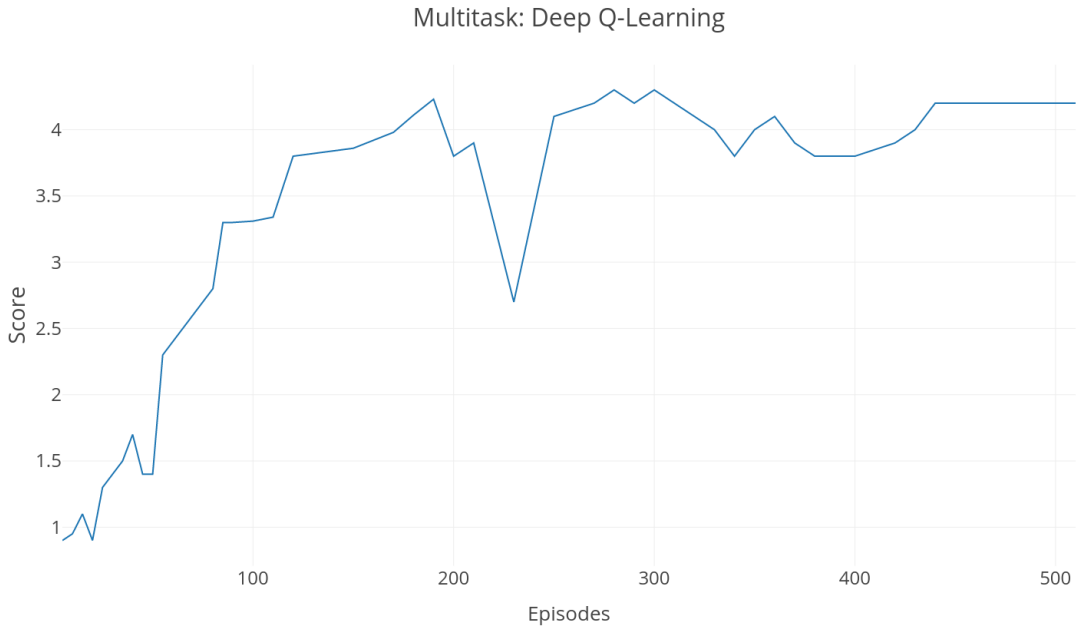

Figure 7: Deep Q-Learning Training

of the game and y-axis score before reaching the terminal state. The agent had troubles adapting to the third phase (see Section 5). Phase 3 is relatively hard to master because it requires the user balance the arrow in the air. At around 230 episodes we saw a drop in score. This is because the network seems to prioritize the first phase of the game. It reached the second phase a few times but was not able to successfully control the paddle for longer periods of time. This is why it stales at approximately 400 episodes. We believe that the network could have performed better with additional training time. It trained for a total of two days. Hopefully, it will be easier to train the network when FlashRL can speed-forward games, see section 7. The results are overall acceptable as we can see that FlashRL deliver quality states that a reinforcement learning agent can learn from.

6 Conclusion

FlashRL offers an easy-to-use architecture for performing RL in Flash-based games. It is demonstrated to work well for Multitask 2, one of the environments included. FlashRL fills the gap that emerged with the deprecation of Flash, Its main focus is RL, but can also be used for other machine learning genres. This paper shows that FlashRL can be used to train RL algorithms, in particular, Multitask 2. The work shows promising results and continuing to expand the game repository may provide new insights about RL in the future.

FlashRL will be kept alive as long as flash environments are an asset to the machine learning community. It is available to the public at `https://github.com/UIA-CAIR/FlashRL`, and can easily be adapted to every research requirement.

7 Future Work

Several improvements are planned for FlashRL. This paper outlined features of the initial version of the FlashRL, and it is by far sufficient for simple reinforcement learning research. As seen in section 5, a Deep Q-Learning based agent can successfully learn from the environment *Multitask* and gradually perform better.

Speed-forward Option

Learning algorithms often require several thousand episodes to gain expert knowledge of the environment. FlashRL is currently limited to the speed of which the game loop is executed (usually 30 fps in real-time). An important improvement would be to lift this restriction and allow algorithms to train at an accelerated rate. This would certainly improve training duration of feedback based algorithms.

Game Repository Analysis

The game repository features many unlabeled, unrated and untested games. Some games are potentially useless in a machine learning setting and require a review. The review phase is time-consuming, and authors of this paper did not have enough time to analyze each of the environments manually. The goal is to add labels and categorize all games in the repository gradually.

Website

A future goal is to allow execution of algorithms from a web interface and to add gamification aspects to the library. This would potentially create competition between researchers much like Kaggle and OpenAI Universe.

Cross-Platform Support

FlashRL is in the initial version, only supported in Python 3 on the Linux platform. The goal is to extend it so that it also can run without modifications on Microsoft Windows operating systems.

References

[1] Marc G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *CoRR* abs/1207.4708 (2012). URL: http://arxiv.org/abs/1207.4708.

[2] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540.

[3] *Flash Testing with Selenium*. Aug. 2017. URL: https://www.guru99.com/flash-testing-selenium.html.

[4] Michael E. Grost et al. "Applications of Artificial Intelligence". In: *CAD/CAM Robotics and Factories of the Future: Volume II: Automation of Design, Analysis and Manufacturing*. Ed. by Birendra Prasad, S. N. Dwivedi, and K. B. Irani. Berlin, Heidelberg: Springer Berlin Heidelberg, 1989, pp. 165–229. ISBN: 978-3-642-52323-6. DOI: 10.1007/978-3-642-52323-6_3. URL: https://doi.org/10.1007/978-3-642-52323-6_3.

[5] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). URL: http://arxiv.org/abs/1509.06461.

[6] Harold L Hunt and II Jon Turney. "Cygwin/X Contributor's Guide". In: (2004).

[7] Yuxi Li. "Deep Reinforcement Learning: An Overview". In: *CoRR* abs/1701.07274 (2017). URL: http://arxiv.org/abs/1701.07274.

[8] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *CoRR* abs/1602.01783 (2016). URL: http://arxiv.org/abs/1602.01783.

[9] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[10] Volodymyr Mnih et al. "Playing Atari With Deep Reinforcement Learning". In: *NIPS Deep Learning Workshop*. 2013.

[11] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). URL: http://arxiv.org/abs/1312.5602.

[12] Matthew Piper. "How to Beat Flappy Bird: A Mixed-Integer Model Predictive Control Approach". PhD thesis. The University of Texas at San Antonio, 2017.

[13] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN: 0136042597, 9780136042594.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[15] Techtonik. *python-vnc-viewer*. https://github.com/techtonik/python-vnc-viewer. 2015.

[16] Ziyu Wang, Nando de Freitas, and Marc Lanctot. "Dueling Network Architectures for Deep Reinforcement Learning". In: *CoRR* abs/1511.06581 (2015). URL: http://arxiv.org/abs/1511.06581.