The PlayStation Reinforcement Learning Environment (PSXLE)

Iñigo Munárriz, Aitor González, Carlos Domínguez
September 2020

1. Abstract

The paper proposes a new way for evaluating Reinforcement Learning algorithms through games using a modified version of the PlayStation 1 emulator that facilitates the use of a simple API in order to expose simple controls and achieve rich game-sate representations. The Playstation Learning Environment (PSXLE) supports the OpenAI Gym interface and the OpenAI Baselines.

2. Introduction

Reinforcement Learning is a form of Machine Learning which uses a system of rewards and evaluates how agents interact with a given environment. Rewards encourage good behaviour and penalize bad ones in order to maximize the notion of cumulative reward. It is one of the three basic Machine Learning paradigms followed by supervised and unsupervised learning.

Agents interact with the environment through actions that are performed based on the **state encoding**, which is the codified representation of the environment that the agent receives in an instant of time. State encoding requires the necessary amount of data the agent will use such as the position, how close enemies are, remaining time on a clock..., if we try to detail too much the environment it can lead to generalization problems.

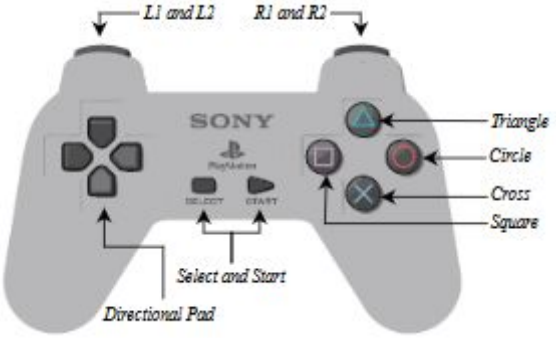
The state encoding allows to predict the next action using, nowadays, deep neural networks like **Deep Q-Networks (DQN)** so agents can interpret even complex states of the environment. Other types of neural networks like, **double DQNs**, prioritised experience replay and Duelling Architectures to enhance the original DQN.

The use of computer games provide many advantages such as the simplicity of the environments and the "**score**" many games usually have, which can act as the reward system.

The goal of this paper is to express how PlayStation environment can prove to be interesting in the **RL** area. Kula World is the example of a possible game to train an agent using the PlayStation modified emulator and the example used in this article.

3. Playstation and Kula World

The Sony PlayStation 1 console is newer than the normally used for RL, Atari-2600. It has 2MB of RAM, 16.6 million displayable colours and 33.9MHz CPU. On the other hand, Atari-2600 has 128B of RAM, 128 displayable colours and 1.19MHz CPU.



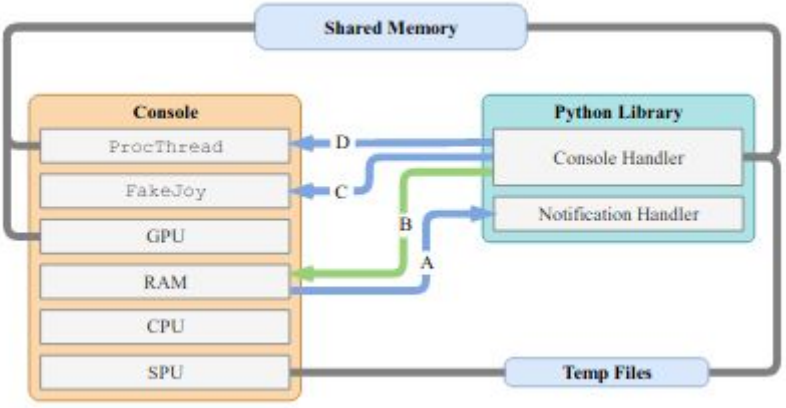
The PlayStation controller has 14 buttons.

Kula world is a game released in 1998 where the player controls a ball around the world. The world consists of a platform where the main objective is to collect coins, fruits and keys with the ball and arrive at the exit platform in order to pass the level.

4. Implementation

The PlayStation Reinforcement Learning Environment (PSXLE) is a toolkit for training agents to play Sony PlayStation 1 games. PSXLE is designed following the standards set by ALE (Arcade Learning Environment) and enabling RL research using more complex environments and state encoding.

The library is built using a fork of PCSX-R, an open source playstation emulator, with a few modifications such as adding a simple Inter-Process Communication (IPC) tools and a Python based console API in order to translate game actions into console functions so it can be used by OpenAI Gym.



A visualisation of the Inter-Process Communication used in PSXLE.

The Console API supports four primary forms of interaction:

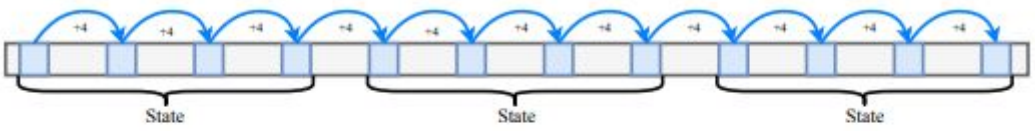
- General:
  - **run** and **kill** control the executing process of the emulator.
  - **freeze** and **unfreeze** will freeze and unfreeze the emulator's execution, respectively.
  - **speed** is a property of **Console** which, when set, will synchronously set the speed of execution of the console, expressed as a percentage relative to default speed.
- Controller:
  - **hold_button** and **release_button** simulate a press down and release of a given controller button referred to here as control events.
  - **touch_button** holds, pauses for a specified amount of time and then releases a button.
  - **delay_button** adds a (millisecond-order) delay between successive control events.

- RAM:
    - **read_bytes** and **write_byte** directly read from and write to console memory.
    - **add_memory_listener** and **clear_memory_listeners** control which parts of the console's memory should have asynchronous listeners attached when the console runs.
    - **sleep_memory_listener** and **wake_memory_listener** tell the console which listeners are active.
- Audio/Visual:
    - **start_recording_audio** and **stop_recording_audio** control when the console should record audio and when it should stop.
    - **get_screen** synchronously returns an np.array of the console's instantaneous visual output.

OpenAI Gym environment uses three game abstraction methods: reset (restarts an episode and returns the initial state), step (takes an action as an argument and its performed in the environment) and render (renders the current state to a window or as texts).

The step function uses an integer which represents the action to perform and returns a tuple containing the **state**, which is the value of the state of the system after performing the action given; **reward**, which gives the reward gained by the performing action; a boolean called **done**, which represents if the level has ended; and **info**, which provides of extra information about the environment. The thing is that OpenAI Gym requires all of these to be returned at the same time. This can be done in two different ways: frame skip and asynchronous run.

The first one works well with simple games, but is not always optimal if different moves require different amounts of time to finish. If the incorrect quantity of frames is skipped a move can be considered as not finished and can lead to problems. Therefore, the second option is used sometimes, which requires a variable which allows us to know when the move is over. PSXLE lets this to be easily implemented using memory listeners that respond to changes in RAM.



(a) A visualisation of frame skip and frame stacking. In [9], only every fourth frame is considered and of those, every four frames are combined (stacked) into a single state representation. New frames are requested by the Python library upon each action, making this approach synchronous.

(b) An asynchronous approach to frame skipping. In environments where actions are long or have variable length, the state transition occurs asynchronously. The transition ends once the immediate effects of the associated action have ceased.

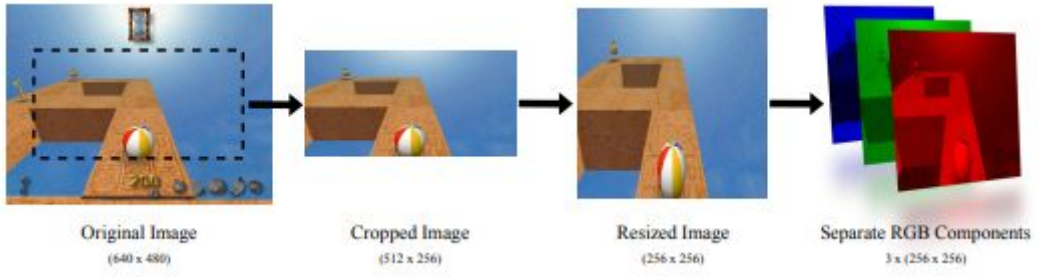
In the particular example of Kula World, there are 4 main actions that players can do: move forward, look right, look left, and jump forward.

- **Move forward:** moves the player 1 square forwards on the platform, in the direction that the camera is facing.
- **Look right:** Rotate the direction of view 90 degrees clockwise, about the line perpendicular to the platform.
- **Look left:** Rotate the direction of view 90 degrees anti-clockwise, about the line perpendicular to the platform.
- **Jump forward:** moves the player 2 squares forwards, over the square in front of it. If the player jumps onto a square that does not exist, the game will end.

The abstraction does not prescribe a state encoding, instead it returns a tuple of relevant data after each move has finished. The contents of the tuple are:

- **visual:** an RGB array.
- **reward:** the value of instantaneous reward resulting from the move that has just been executed.
- **playing:** a value indicating whether the player is still ‘alive’.
- **clock:** the number of seconds that remain in which the player must complete the level.
- **sound:** None, an array of Mel-Frequency Cepstral Coefficients (MFCCs) or an array describing the raw audio output of the console.
- **duration_real:** the amount of time the move took to complete.
- **duration_game:** the amount of the player’s remaining time that the move took to complete, relative to the in-game clock.
- **score:** the score that the player has achieved so far in the current episode.

The RGB array is processed this way:



5. Rewards

RL needs a rewards system, which will lead the agent to desirable behaviour by adding numerical rewards to the action that an agent performs. In the end, the reward is just like a score number which will increase if the agent performs well.

Any event that leads to a possible loss should be punished as undesired behaviour. In Kula World you can lose by falling off the edge of the platform, being ‘spiked’ by an object in the game or running out of time.

For these reasons, the remaining amount of time is also considered by the agent in order to choose an action and avoiding it to lose time. If the time that remains from a level is little, the agent will understand that moving will cost time too, and that way, the agent will end learning to use only the necessary moves.

A function is specified for the reward system: "**score_to_reward**", which will reward based on the amount of score an action done gives: coins give 250, collecting a key 1000, … (see the table below for all the actions)

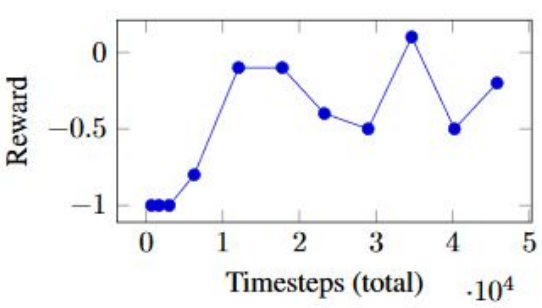
Non-scoring action will be considered typically as a little negative score value and the sum of the rewards are limited in order to provide stability in learning.

Reward function		
Event	Score change	Reward
Coin collect	+250	0.2
Key collect	+1000	0.4
Fruit collect	+2500	0.6
Win level	-	+1
Lose level	-	-1

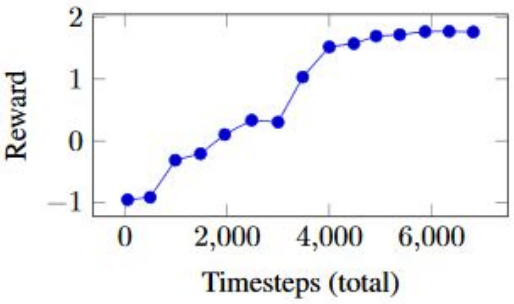
6. Evaluation

The Kula-v1 Gym environment supports Kula World firsts 10 levels. It uses the console's screen as its state encoding and (by default) uses the reward function described in table above, at section 5-Rewards. This environment was used with deepq and ppo2 from OpenAI Baselines. This approach to training is not particularly interesting, since the agent starts at the same situation in each episode. As a result, the agent simply learns how to beat that level in the shortest time with the highest score. In order to avoid this problem additional starting positions are created using the save state option.

A simple DQN agent was implemented, for use with Kula-random-v1. The agents were trained on Level 1 of Kula World using a visual state encoding. The results are summarised in the graphs below.

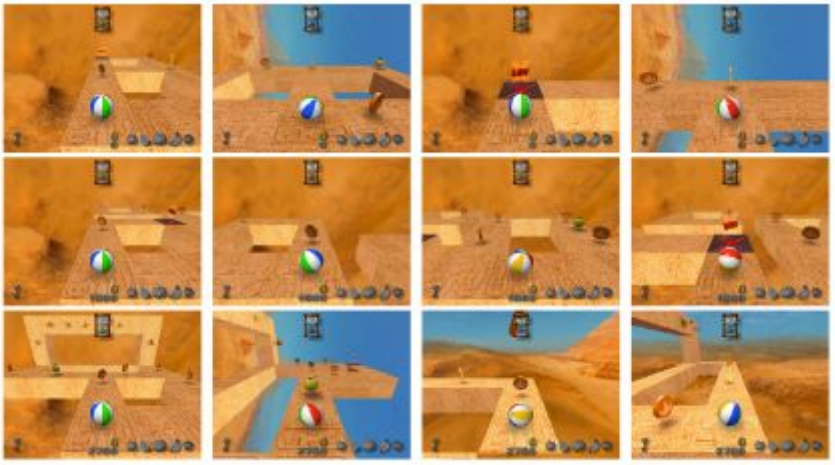


(a) The deepq baseline.

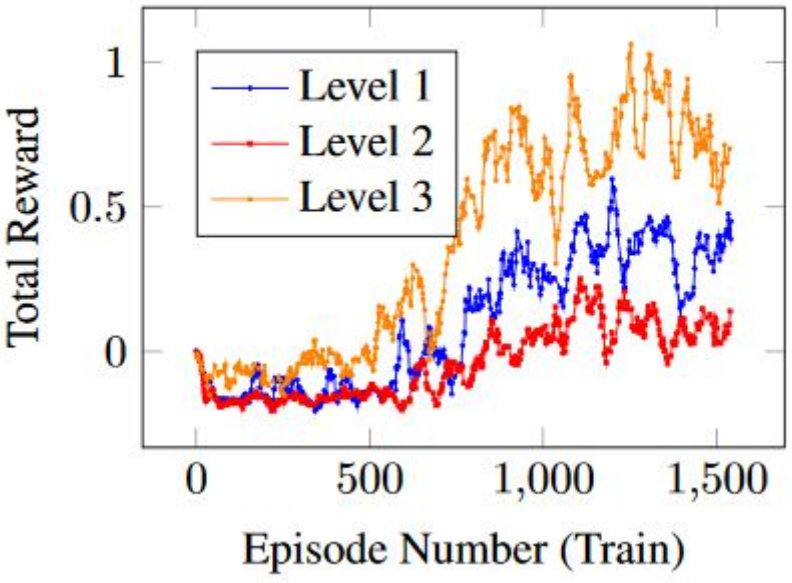


(b) The ppo2 baseline.

The environment kula-random-v1 presents the agent with one of four different starting positions for the first three levels of the game, randomly selected. There is an extra starting point at level 2 that is reserved for use in validation episodes. We can see the starting points at the Figure below.



On level 2, you have to make a decision, jump to another platform or not, doing it you can obtain more coins, but trying it you can lose the game. The level 3 introduces you to new physics that apparently are not intuitive to humans. Due to that, humans normally obtain better marks at the level 2. Instead, the agent was most successful in learning how to play level 3. On the next graph we can see the agent proficiency on each level during training.



Another environment was developed using audio: Kula-audio-v1. The state encoding is the richest yet, containing visual output, audio output, time remaining on the level and score. MFCCs can be used in order to represent the audio output of a move.

Finally, since the agent starts each episode from a specified state, the states in its memory are dominated by those that occur at the start of the level. For PlayStation games, which are usually quite complex and have long levels, this can result in lengthy training times. State replay is proposed as an alternative in which an agent can resume the game from a state which it has previously visited, allowing the agent to perform a different action. This can be trivially implemented using the save_state and load_state methods in PSXLE and could help to reduce the time required to learn complex games. No agents have been trained using this approach.

7. Conclusions

Two different algorithms of RL such as DeepQ and PPO2 from the OpenAI baselines show that they perform vastly differently in Kula World. This approach can be used with any other game as the design of PSXLE simplifies the build of abstraction which supports interfaces such as OpenAI Gym and therefore we could say that games can become optimal environments in which RL algorithms effectiveness can be evaluated and where new environments can be built in the future, with more complex and richer state spaces.

8. Bibliography

- https://arxiv.org/pdf/1912.06101.pdf
- carlospurves/psxle: A Python interface to the Sony PlayStation console.
- Reinforcement learning - Wikipedia