ISSN 2278-3091
Volume 11, No.5, September - October 2022
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse011152022.pdf
https://doi.org/10.30534/ijatcse/2022/011152022

Exploring the Integration of Lisp into a Modern Reinforcement Learning Project Through the Use of Hy

Kuo-pao Yang[1], Max Cole[2], Hayden Israel[3], Madison Leblanc[4], Noah Vernon[5]
[1,2,3,4,5]Computer Science Department
Southeastern Louisiana University
Hammond, LA 70402 USA

Received Date: August 14, 2022 Accepted Date: September 20, 2022 Published Date: October 06, 2022

ABSTRACT

This paper explores the usage of Lisp in a small modern Reinforcement Learning (RL) project. The Lisp dialect, Hy programming language, is used to incorporate the traditional libraries and packages in up-to-date workflows. This project is centered around the usage of NetHack for RL. The MiniHack sandbox framework and NetHack Learning Environment (NLE) are used to create custom training/testing environments and tasks. The MiniHack sandbox framework creates a simple level editor and creation interface for use in the training and evaluation process of the agent. NLE is chosen as the working environment. For the agent model, this project adopts Torchbeast's PolyBeast, a PyTorch implementation of the IMPALA architecture. The usage of Hy within this project is forefront, and so it is implemented as much as possible to accomplish the tasks.

Key words: Hy, Lisp, NetHack, Reinforcement Learning.

1. INTRODUCTION

This project explores the use of Lisp within a small Reinforcement Learning (RL) development and examines its use within a modern scenario. Lisp [1] is the second-oldest programming and was favored for use in Artificial Intelligence (AI) research. Over time, Lisp has fallen out of use in the field of AI and is really used in modern day tasks. The topic focuses on exploring the use of RL within a roguelike, a specific genre of game. Traditional roguelikes are turn-based, grid-based games based on high-complex randomly generated environments with "permadeath" mechanics, meaning the users start completely over if they die [2]. These traits combine together to make an interesting and an effective learning environment for RL.

There are a few implementations of these environments, such as one for the original Rogue [3], Dungeon Crawl Stone Soup [4], and NetHack [5]. The NetHack implementation is a better fit of the project's needs. The NetHack Learning Environment (NLE) is chosen as the working environment. NLE provides a RL interface to the game NetHack [6] along with providing tasks for the agents. Additionally, the sandbox framework MiniHack [7] is used. It can communicate with NLE to create custom training/testing environments and tasks.

This project adopts a Lisp dialect called Hy [8] in order to utilize NLE and MiniHack. Hy programming language is a Lisp dialect embedded in Python, allowing Lisp code to compile into Python. By using Hy, it makes available to utilize Lisp code while incorporating Python libraries into the development workflow. This project allows not only to use MiniHack and NLE, but also to take advantage of useful libraries such as PyTorch, Pandas, Numpy, and Tensorflow. It creates a simple level editor with Hy programming language by manipulating the functionality given by MiniHack level generator. To handle the RL agent, this project is able to take advantage of Torchbeast's PolyBeast [9], which when coupled with the des file reading of MiniHack, completed the goal.

In summary, this paper explores the use of the Lisp dialect Hy and evaluates its capabilities within a modern-day RL task. This project accomplishes the task by using Hy to build a small application that takes advantage of MiniHack level generator in order to create environments for the agent. To train and evaluate the agent, this project manipulates the functionality provided by TorchBeast. Finally, conclusions are presented regarding the use of Hy and issues for future research are discussed.

2. BACKGROUND

2.1 Lisp and Hy Programming Languages

Lisp, which is derived from "LISt Processor" [10], is a programming language developed in 1958. It is known for its fully parenthesized prefix notation and developing some of the foundational ideas of computer science such as tree data structures, automatic storage management, read-eval-print loop, conditionals, dynamic typing, higher-order functions, and recursion. Because Lisp's source code is made from lists, this gives rise to its notable metaprogramming capability. When Lisp was first released, it had notable use in Artificial Intelligence (AI) research almost immediately. However, slowly over time, its presence has dwindled within the AI space.

Despite the dwindling popularity of Lisp, it experienced the creation of several new dialects over time, such as Clojure, Scheme, Common Lisp, and more. One notable dialect, the focus of this paper, is Hy. Hy programming language enables the translation of Lisp's s-expressions into Python's abstract syntax tree. This allows Python libraries to be imported and used alongside Hy code. The access to Python libraries is crucial, as many of the AI related libraries for Lisp are able to be integrated into modern RL projects through the use of Hy programming language.

2.2 Reinforcement Learning and NetHack

Reinforcement Learning (RL) requires complex and challenging environments to test in. The genre of games called roguelikes have been found to work very well for RL. Roguelikes are typically very difficult games, involving procedurally generated complex mechanisms and environments. These mechanics coupled with the resource management, exploration, and dynamic nature of the environments within, make it an interesting method of testing for RL.

One of these games that has garnered more attention than others is NetHack [11] [12]. NetHack is an open-source roguelike released in 1987 and features many things prominent to the roguelike genera. Its features procedurally generate environments, monsters, dungeons, and treasures. The gameplay consists of a player first choosing or randomly selecting the character's race, role, sex, and alignment. The player must then seek out the "Amulet of Yendor" and complete the numerous "side-quests" to beat the game. The complex nature of the game and possibilities that arise from the dynamic elements earned its popularity in both gaming and RL.

One specific NetHack environment is the NetHack Learning Environment (NLE). This environment provides a traditional style RL interface around the terminal interface from NetHack. The NLE environment also supports actions corresponding to all the ones available to the player along with various observation spaces for the agent. An OpenAI [13] Gym interface is also offered along with predefined tasks, reward functions, action spaces, and a dashboard for evaluations. In addition to the pre-defined tasks, NLE provides easy access to create additional tasks.

3. IMPLEMENTATION

3.1 MiniHack

Since NLE environment focuses on the full game of NetHack, it is difficult to test certain conditions and specific scenarios that the practitioner would want. To fix this issue, the MiniHack sandbox framework scales the world down into scenarios that are more suited for testing and evaluating the desired conditions. Like NLE, MiniHack provides a significant number of challenging tasks, but the primary focus is on streamlining the task creation process and increasingly scale the complexity. To accomplish the task, MiniHack leverages the use of the des-file format that describes the levels within NetHack. This format allows creations to be made as the underlying language is complex. MiniHack simplifies the level creation through the usage of its level generator. The level generator provides functions to easily add objects, monsters, environmental features, and general map elements. The get_des function then allows the direct export of the generated elements to a des file. Figure 1 shows an example of a generated des file for the level "Quest Hard." Additionally, MiniHack provides the reward manager to streamline the goal creation process.

```
# Inspired from Baalzebub level of NetHack
MAZE: "mylevel", ' '
FLAGS: noteleport, corrmaze
GEOMETRY: right,center

MAP
--------------------------------------------------
|                 ----              ----
|        ----     |    ----------- |
| ------      | ---------|.........|--|
| |....|  -------|.LLLLL..------------------
---....|--|........LLLLL.....................|----
+.........+........LLLLL.....................|
---....|--|........LLLLL.....................|----
| |....|  -------|.LLLLL..------------------
| ------      | ---------|.........|--|
|        ----     |    ----------- |
|                 ----              ----
--------------------------------------------------
ENDMAP
```

Figure 1: Example of Generated des File

3.2 Level Editor

To add a level of complexity, this project creates an interface that allows users to customize the agent's overall environment that the agent is placed in. It provides a great level of customizable variety in each scenario that the agent experiences, rather than having it interact with the same exact scenario from time and time again. This is primarily accomplished using the level generator functions within MiniHack, as demonstrated in Figure 2.

This project keeps the interface itself simple for ease-of-use shown in Figure 3. It provides several buttons that each affect the agent's environment in different ways. The "Add Monster" will place a completely random creature into the level, which may very well be hostile towards the agent. The "Add Trap" will add a teleportation trap in a random spot within the environment, which will teleport the agent to a random location should it step onto the trap. The "Add Object" will add a completely random object that the AI can also pick up and use. The "Add Boulder" will place a boulder within the level that the agent can push around wherever it wants. Finally, the "Submit" button will save all changes to the level.

```
(setv lvl_gen (minihack.LevelGenerator :w 10 :h 10))

; Events when the user selects a button
(while True
(setv [event value] (.read Window))
(if (or (= event sg.WIN_CLOSED) (= event "Submit"))
  (break)
  (setv des (.get_des lvl_gen)))

(if (= event "Add Monster")
  (.add_monster lvl_gen)
  )

(if (= event "Add Trap")
  (.add_trap lvl_gen)
  )

(if (= event "Add Object")
  (.add_object lvl_gen)
  )

(if (= event "Add Altar")
  (.add_altar lvl_gen)
  )

(if (= event "Add Boulder")
  (.add_boulder lvl_gen)
  `
```

Figure 2: Code for Level Generator Functions

The users can press each of the "Add" buttons as many times as they would like to add even more of whatever they wish. For example, seven completely random monsters will be added into the environment if the users are to press the "Add Monster" button seven times. If the users are to press the "Add Boulder" four times, four boulders will appear in random locations within the level. The same concept applies for all of the other buttons. The "Submit" button saves the level and closes the interface.

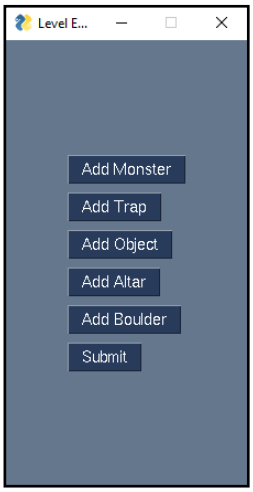

Figure 3: Menu for Level Generator

Once the users are satisfied with their selections and submit their changes to the level, the console will display a preview of the level and everything that is added to it. In Figure 4, the users select the "Add Monster" button two times, and the "Add Object" button once. The result creates a level that will feature two random creatures and one random object. When the users preview the level, they can see the changes.

```
MAZE: "mylevel", ' '
FLAGS:hardfloor
INIT_MAP: solidfill,' '
GEOMETRY:center,center
MAP
..........
..........
..........
..........
..........
..........
..........
..........
..........
..........
ENDMAP
REGION:(0,0,10,10),lit,"ordinary"
MONSTER:random,random
MONSTER:random,random
OBJECT:random,random
TERRAIN:rect (0,0,9,9),'L'
```

Figure 4: Level Generator Results

Figure 5 shows a visual of the created level using a tile set for better visual representation. There are two monsters and one object in the room selected by the user.

Figure 5: Visual of Level Generator Using Tile Set

3.3 Adding an agent

Once the interface is created, the project implements an agent to train with the new environments and tasks. Since it uses Hy programming language and has access to Python libraries, such as MiniHack and NLE, this project takes advantage of Hy programming language and uses TorchBeast. TorchBeast provides a RL platform within PyTorch and implements a version of the Importance Weighted Actor Learner Architecture (IMPALA) [14] algorithm. The IMPALA is a distributed agent that uses resources efficiently with single-machine training while also allowing the capabilities to scale to thousands of machines without sacrificing resources or efficiency. TorchBeast is designed with a pure Python approach in mind, along with providing OpenAI interface. This project implements with the C++ and Python Variant, PolyBeast, for its higher performance potential over the MonoBeast version. In addition to the

capabilities of TorchBeast, this project also choses it since MiniHack has a direct implementation of PolyBeast already integrated within, thus easing the burden of integration.

4. DISSUSSION

This project encountered issues between Hy and MiniHack. After each module was implemented, we began the task of integrating the level editor with the TorchBeast agent and MiniHack environments. We started facing numerous issues with Hy programming language and its interactions between the various packages and frameworks. The most notable issue with the level editor was the creation of the OpenAI environment. In order to use an environment created in OpenAI's Gym interface, the users must first register the environment and set parameters. Since MiniHack creates environments by leveraging the use of des files, a des file must be given when registering an environment for OpenAI. When Hy generates the des file from MiniHack level generator, the conversion of Hy to Python did not function correctly, and thus could not be passed as an argument in the functions. To work around this situation, we resorted to using Python and creating a separate environment class registered with that. We then, placed this environment within the MiniHack package's environments so that we could easily use the scripts supplied with the custom environment. Since we could directly pass the des file data as an argument to the custom environment, we exported it to a dat file directly and then created a separate statement within the class to load the dat file.

In addition to the issues with Hy and MiniHack, this project also encountered severe problems with the simple level editor GUI. PySimpleGUI [15], a package to transform tkinter, Qt, WxPython, and Remi GUIs into simpler interfaces, has numerous issues with Hy. The most difficult one to deal with was simply providing output to the terminal once the interface was interacted with. If any output to the terminal was inside the interface, Hy would fail to run and present end-of-file statements. We also encountered other issues such as GUI interface not always having their states saved. This project successfully solves Hy and GUI problems by resorting to other means of outputting to the terminal, primarily by using the altered des file as the output.

5. CONCLUSION

By using Hy, this project makes it possible to utilize Lisp code while incorporating Python libraries into the development workflow. Despite the multitude of issues, it is not only able to successfully implement the small Hy based level editor for MiniHack and integrate it with the TorchBeast's PolyBeast agent, but also tests the usage of Hy with some modern-day toolkits and scenarios. This project successfully accomplishes and evaluates the goal naturally, rather than forcing the usage when it would have been proven cumbersome. Overall, the usage of Hy is proved useful along with the overall syntax and style being enjoyable to write. However, the compatibility of certain packages and general lack of documentation on any issues that arise causes massive challenges with the workflow. Using Hy is an interesting combination of Lisp and Python, it can be used in a modern-day practitioner's toolkit. In the future, this project would like to explore the usage of the RLlib library in an effort to see if it works better with Hy as well as to expand the Level Editor application into something bigger of use.

REFERENCES

1. J. McCarthy, **Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I**, *Communications of the ACM*, vol. 3, no. 4, pp. 184-195, April 1960, doi: 10.1145/367177.367199
2. IRDC, **Berlin Interpretation**, *International Roguelike Development Conference*, September 2008, Available: http://www.roguebasin.com/index.php?title=Berlin_Interpretation, Accessed: September 2022.
3. A. Asperti, D. Cortesi, C. De Pieri, G. Pedrini, and F. Sovrano, **Crawling in Rogue's Dungeons With Deep Reinforcement Techniques**, *IEEE Transactions on Games*, vol. 12, no. 2, pp. 177-186, June 2020, DOI: 10.1109/TG.2019.2899159.
4. D. Dannenhauer, M. Floyd, J. Decker, and D. Aha, **Dungeon Crawl Stone Soup as an Evaluation Domain for Artificial Intelligence**, *AAAI-19 Workshop on Games and Simulations for Artificial Intelligence*, Hawaii, 2019, arXiv:1902.01769.
5. H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel, **The NetHack Learning Environment**, *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vancouver, Canada, pp. 1-28, December 2020, arXiv:2006.13760.
6. NetHack, **NetHack Version 3.6.6**, Available: https://www.nethack.org, Accessed: September 2022.
7. M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Küttler, E. Grefenstette, and T. Rocktäschel, **MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research**, *35th Conference on Neural Information Processing Systems (NeurIPS 2021)*, pp. 1-33, December 2021, arXiv:2109.13202.
8. Hy, **Hy Society**, Available: https://github.com/hylang/hy, Accessed: September 2022.
9. H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette, **TorchBeast: A PyTorch Platform for Distributed RL**, pp. 1-10, October 2019, arXiv:1910.03552.
10. R. Jones, C. Stewart, I. Stewart, **The Art of Lisp Programming**, Springer Science & Business Media, December 1989, ISBN-13: 978-3540195689.
11. SYNCED, **NetHack: Fast & Complex Learning Environment For Testing RL Agent Robustness & Generalization**, July 2020, Available: https://syncedreview.com/2020/07/01/nethack-fast-comple

x-learning-environment-for-testing-rl-agent-robustness-generalization, Accessed: September 2022.
12. Shen, **NetHack-inventory**, April 2006, Available: https://en.wikipedia.org/wiki/File:Nethack-inventory.png, Accessed: September 2022.
13. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zarembam, **OpenAI Gym**, pp. 1-4, June 2016, arXiv:1606.01540.
14. L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, K. Kavukcuoglu, **IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures**, June 2018, arXiv:1802.01561.
15. PySimpleGUI, **Python GUIs for Humans**, Available: https://pysimplegui.readthedocs.io/en/latest, Accessed: September 2022.