Object Detectors as Input for Reinforcement Learning Agents

Benjamin van Oostendorp
DigiPen Institute of Technology
Redmond, WA, United States of America
ben.vanoostendorp@digipen.edu

***Abstract*—In traditional reinforcement learning applications with images as input, the observation for the agent to learn from, is an image. In these models, a Convolutional Neural Network (CNN) is typically used to extract the features before for the learning process, in order to maximize the cumulative reward. In this paper, a different approach for pre-processing the input for reinforcement learning agents is considered. The proposed approach uses object detectors instead instead of CNNs, and converts each input image into bounding boxes and object locations for the agent to learn from.**

***Index Terms*—object detection, reinforcement learning, deep q-learning**

I. Introduction

In reinforcement learning, generalization of agents and models allows an agent to perform similar tasks in a different environment. Often, the lack of generalization capability prevents tuned agents from being able to complete similar tasks in different environments, or even similar tasks in slightly different environments [1]. Often, AI agents have to be made specifically for the environment and tuned accordingly, even for environments that are very similar. Other times, the new environments that an agent is proposed to learn, doesn't have a direct link to the agent. For example, in self driving cars, the driver of the car does not have direct access to the physics of the universe, or in video games, the player doesn't have all access to the memory or debug tools.

Object detectors [2] can handle these problems, since rules can be derived from bounding boxes based on human experience in the environment. This allows agents to extend to environments where reward functions are not provided. This approach also treats observations similar to humans; instead of seeing every individual pixel in an image, it focuses on the specific objects within the image. This method may promote quicker learning as there is an inherent amount of information embedded in the observation.

Another potential advantage to this approach is explainability. Traditionally, it is very difficult to interpret why a Convolutional Neural Network (CNN) makes the decisions it does. While this approach does not currently attempt to explain the actions of the agent, using techniques such as mutual-information-guided linguistic annotations (MILAN) [3] or fuzzy-based techniques could lead us to understand the reasoning behind the agent's actions.

In this paper, first the environments are introduced, then the implementation is analyzed, and finally the results and viability of the method are discussed.

II. Methodology

While there is a lot of research on classic reinforcement learning environments, the goal of this approach is to be able to tackle more complex problems with the ability to create logical rules for the system from objects in the scene. The games of choice for the proposed agent to learn are Shovel Knight: Shovel of Hope [4] (referred to as "Shovel Knight") and Super Mario Bros.: The Lost Levels [5] (referred to as "Mario").

Shovel Knight is selected as it appeared as an interesting task since there are no methods to gain direct access to its internal variables. The only information from the game that can be gathered is images as frames from game play. It poses an unique challenge of needing to create a custom reward function as well as not being directly accessible through the environment. Traditionally, the environment is created to be directly controlled, like in OpenAI's Gym [6]. In these environments, the agents can directly interact and the environment "waits" for the agent to take actions, pausing simulation until moving to the next state. Shovel Knight is a more traditional game in the sense that a player would interact with the game via a keyboard or controller, process the action within the game, and display the next state for the player to interpret.

Mario, however, has a OpenAI gym implementation [7] but is a quite complicated video game and has not been the subject of much exploration in reinforcement learning.

Training data for the object detector is collected by capturing screenshots every few seconds, and then labeling bounding boxes and classes manually using MakeSense.ai [8]. The custom training data set for the proposed approach was created as part of this project.

A. Object Detection

There are many object detectors and classifiers in the literature, such as Region-based CNN (RCNN) [2], Spatial Pyramid Pooling (SPP-Net) [9], Single Shot Detector (SSD) [10], and You Only Look Once (YOLO) [11]. Each of these detectors complete two tasks: finding an arbitrary number of objects in a given image, and then classify each object and estimate its position and size with a bounding box. For the

Cite as: B. van Oostendorp, "Object Detection for Reinforcement Learning Agents", *Syst. Theor. Control Comput. J.*, vol. 3, no. 2, pp. 9–14, Dec. 2023.
DOI: 10.52846/stccj.2023.3.2.51

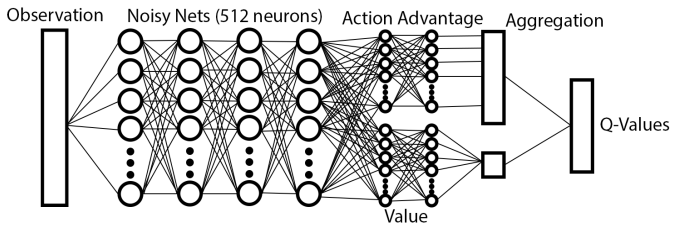

Fig. 1. RAINBOW Agent Architecture

proposed framework, YOLOv5 [12] is adopted. The main reason for choosing YOLOv5 is that YOLOR [13] was not available, and after analyzing several models such as RCNN, Fast RCNN [14], Faster RCNN [15], and YOLOv4 [16], YOLOv5 outperformed the other models in terms of accuracy and inference time.

The specific version of YOLOv5 used is the YOLOv5l, the second largest version of the full YOLOv5 algorithm. This version handles images that are 640x640 in size, and has an average inference time of 460 milliseconds. This version is chosen as the accuracy and speed were both adequate. A different network configuration could be chosen for faster inference times or to handle images of different sizes.

B. Agent

The reinforcement learning agent (referred to as "the agent") is built using PyTorch [17]. The agent is a an implementation of the RAINBOW agent [18], consisting of a double Q-learning architecture [19], a prioritized replay [20], dueling networks [21], multi-step learning [22], distributional reinforcement learning [23], and noisy nets [24]. Each network consists of 4 noisy layers consisting of 512 neurons each with 2 noisy layers consisting of 512 neurons for both the value and action advantage portion of the dueling network, displayed in Fig. 1. Each layer uses the ReLU activation function and the final output uses the Softmax activation function. The optimizer is PyTorch's implementation of ADAM [25], and the loss calculation is Mean Squared Error.

In order to compare the proposed method to the more traditional CNN Deep Q-learning environment, both the proposed approach and the CNN-based approach were implemented in the Mario environment. For the CNN-based Mario implementation, the CNN architecture, nicknamed the "NatureCNN" based on the architecture from [26], is used in place of YOLOv5. This is useful so the methods can be compared.

C. Action

Since Shovel Knight does not allow direct access into the game, an outside method to take actions must be employed. This is done by creating a virtual controller with vgamepad [27]. Vgamepad works by using the ViGEM C++ framework [28] with Python bindings, allowing the agent to take actions within Shovel Knight.

Mario does contain an OpenAI gym interface, so the process of obtaining states and taking actions is handled within the implementation of the environment itself.

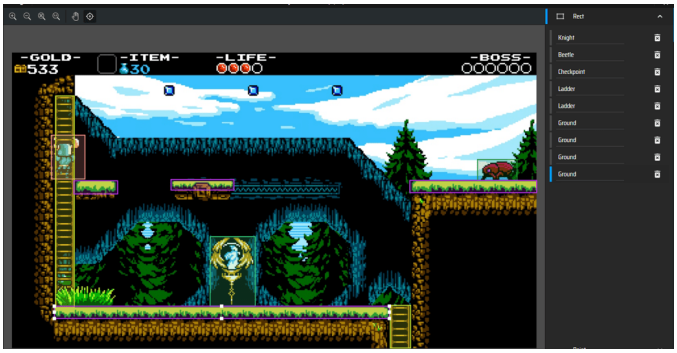

Fig. 2. Makesense.ai data creation

III. Implementation

A. Object Detection Training

The YOLOv5 portion of the framework is trained using screenshots taken from the game, manually labeled with Make-Sense.ai, as shown in Fig. 2, and exported in COCO format [29]. The whole data set consists of 121 images processed this way with images being captured throughout the first level in Shovel Knight. The YOLOv5 network is then trained first and separately from the agent until the accuracy of each class was at least 80

For Mario, the same method is used to train the YOLOv5 network, but with 70 images instead. The task of manually labelling training data can be a considerable time investment, therefore a balance between time-commitment and accuracy has been considered.

B. Environment Observation Handling

The first part of handling the observation is getting the images during game-play in Shovel Knight. The approach used was obtaining screenshots of the game using mss [30], a Python package that captures screenshots and saves them as PNG images. Next, the image must be reduced in size in order to be processed by the YOLOv5 network, from 1280x720 to 640x320. This is done with openCV [31] by down-scaling the image with nearest-neighbor interpolation. With the new image resized, the image is then letter-boxed, putting black bars around the image, to make it a square 640x640 image to match the required input size. Next, the image is passed to the Graphics Processing Unit (GPU) so that it is on the same device as the agent. The image is then converted to floating point numbers, normalized between 0 and 1, and an extra dimension is added to match the input requirements of YOLOv5. Finally, the adjusted image is passed through YOLOv5 and Non-Maximum Suppression (NMS) [32] to receive the required inferences. The layout of each object in the inference is the left, top, right, and bottom positions of the bounding box, followed by the confidence and class of the object, as shown in Fig. 3.

Once inferences are obtained, they must be transformed into a format that the agent will understand. In this implementation, the agent is allowed to know the locations of 150 objects.

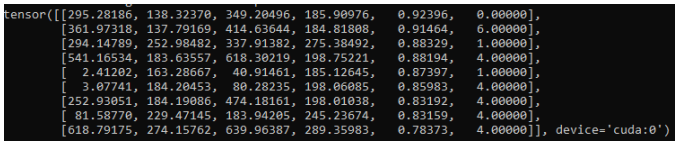

Fig. 3. Example of bounding box data

Fig. 4. Example of object ordering

This number can be tuned to fit different environments. The first step in this process is to sort all of the objects from left to right, then bottom to top. This is done to keep objects in similar positions as they were previously shown in. Next, all of the objects are extracted from the inferences. This is done by extracting objects based on class, and then discard the class and confidence value for the object. For each class, the bounding boxes are flattened and then stored in pre-determined locations of the observation array, as shown in Fig. 4. This is done such that objects of the same class always appear in the same location in the observation for every state. Once all of the classes have been moved into the observation array, it is normalized between 0 and 1, as the object detector's inferences are pixel locations rather than relative image locations. This is done for both Shovel Knight and Mario object detection environments.

The final step in creating the observation for Shovel Knight is calculating the health of the agent. The agent's health is detected directly by interpreting pixel values at the location where health is displayed in the game. The health is is then augmented onto the bottom of the observation array, and normalized between 0 and 1.

C. Action Handling

Now that the observation is constructed for Shovel Knight, it gets passed to the agent to output the action to take. The agent computes the Q-values of the observation for each action, and returns the index of the action with the maximum Q-value. This action is an integer value between 0 and the count of all possible actions, and must be interpreted into controller inputs. This is done by using a bit-wise AND for each button the agent can press. Each button acts as a mask of the action passed in. If the bit is set, the button is pressed, and if it is not set, the button is released. After every button is parsed from the action, the state of the virtual controller is updated.

D. Reward Calculation

One of the most difficult portions of handling environments that do not already have reward functions is writing a custom reward function. One major benefit to this approach is human-like rules can be generated for the environment. As a player of the game, relationships between objects in the image can be used to generate a reward function, and methods like reward shaping [33] can be used to improve upon the reward function.

Fig. 5. Reward calculation example for Shovel Knight

For the case of Shovel Knight, progression is derived by movement and health. The first basic check is if the Knight is dead, and if so, return a large negative reward. If the Knight is not dead, but it is not found in the current frame, use the health gain/loss as a reward. Then, one of two things is done based on if the Knight is in the center of the screen or off to the sides. In Shovel Knight, the camera scrolls/follows along with the Knight, usually keeping it in the center. However, there are some cases where this is not true. When the Knight is not centered, calculating progression is trivial as the movement can be calculated by subtracting the previous location of the Knight from the current position. In the case where the Knight is centered in the frame, two candidate objects are found. Candidate objects are two objects with similar size and similar position. The tolerance for the difference in position of these objects is relatively small, but non-zero. Once two objects have been found, the difference between the current and previous object's position is used to determine movement.

In Fig. 5 the process of object tracking is shown to determine progression. On the left is the previous frame, and on the right is the current frame. The sand block, outlined with a red box, is found to be of similar size and in a similar position, and the progress the agent makes is the difference between the current sand block's location and the previous sand block's location.

The final portion of the reward calculation is to compute the weighted sum of the progression and health with tunable parameters. The current implementation uses 30% of the health reward and 70% of the progression reward.

The reward calculation for Mario is provided within the implementation of the gym environment, based on memory locations for Mario to determine forward or reverse progression. This reward is then normalized between -1 and 1.

IV. Results

The agent for Shovel Knight is trained for 1000 episodes consisting of a maximum of 1000 frames each, with the reward for each episode being recorded. Upon reaching the end of an episode or dying, the agent resets the environment to the beginning of the first stage, and the next episode begins.

The agent for Mario was trained for 1 million frames. Each episode is determined from the rules inside the Mario environment, where each level is given a number of time units, and once those time units run out, Mario dies. The agent is given three lives (unless earning more), and the episode is over

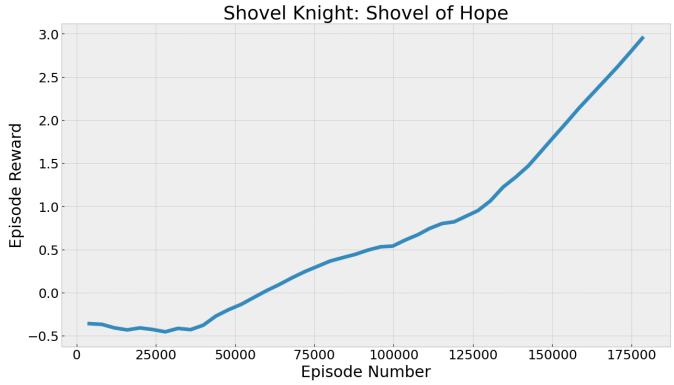

Fig. 6. Reward per episode for Shovel Knight

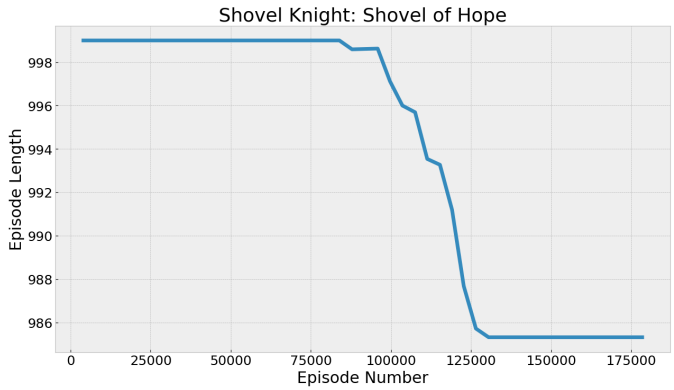

Fig. 7. Episode length for Shovel Knight

once Mario runs out of lives. Both the traditional approach and object-detector approach is trained and compared, with the traditional agent utilizing RAINBOW agent with a CNN as a feature extractor.

V. Discussion

To show the learning of the agents, the rolling mean reward over 10 episodes is collected as well as the length of each episode.

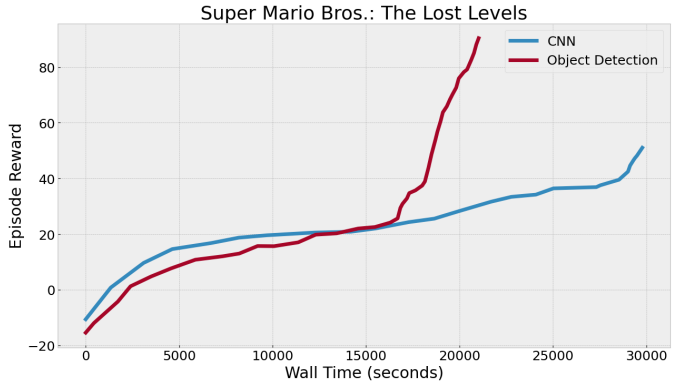

Fig. 8. Reward vs wall time for Mario

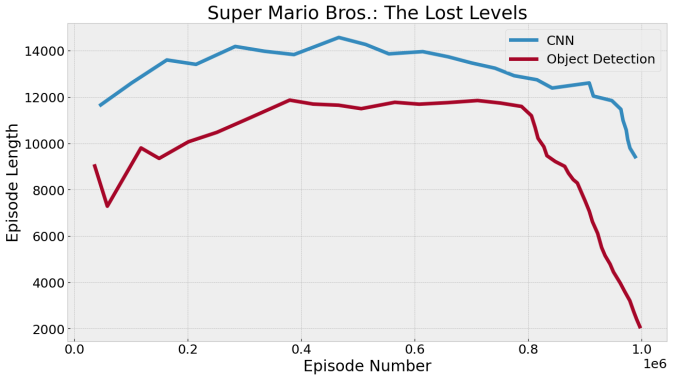

Fig. 9. Episode length for Mario

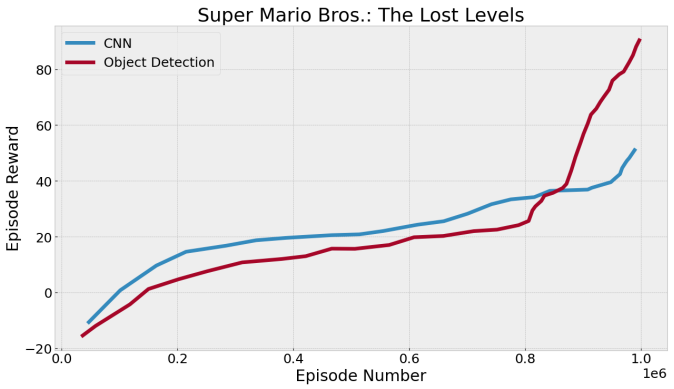

Fig. 10. Reward per episode for Mario

From Fig. 6 it can be seen that over the course of training, the agent is learning the environment with a steadily increasing reward, and the length of each episode is slightly decreasing as shown in Fig. 7. This hints that the agent getting further progress in each episode, and encountering new problems it has not yet faced and not having the experience of surpassing them. Granted, the length of these episodes does not decrease by a large magnitude. Better metrics such as distance covered and health over time in the episode will be tracked in future work to better determine training of the agent.

For Mario, there is a pretty distinct picture of the object-based method reaching a moment where it begins to learn the environment much quicker. From Fig. 10, for the first 800,000 episodes, it is learning with about the same rate as the traditional agent, but obtaining a slightly lower reward. This indicates that the agent is more than likely to die earlier in the episode than the traditional method, as hinted at in Fig. 9. Then, around episode 800,000 it begins to rapidly gain more reward than the CNN-based method. This could be due to the fact that there is inherent information stored inside the observation, so the network does not have to account for positions of filtered items as in the CNN method. This may allow for faster relationship learning.

The final comparison between the object-based and CNN-based methods is the computation time, as illustrated in Fig.

8. Because the object detection architecture was chosen for speed, there is a large gap in how long each agent took to train. The object-based agent took 5.8 hours, compared to 8.3 hours for the CNN-based agent. This is entirely due to training an additional CNN on top of the DQN will decrease the speed of inferring, but it may aid in mastery of the environment. Future work could help determine if this is the case.

One important question that remains, is what would the performance be after another million or more frames of training on the Mario environment, as this example might be indicating overfitting of the object detector agent. Perhaps both agents will come to learn the environments with a similar level of mastery, or one may outperform the other. This will be addressed in a future work.

VI. Conclusion

In this paper, a new approach to processing the inputs for reinforcement learning agents is proposed, and demonstrated in applications in 2 video games: Shovel Knight and Mario. It has been shown that this new approach is capable of learning the given environments, and in the case of Mario, can outperform the traditional CNN architecture of environments with image inputs. This form of observation handling may allow agents to learn in environments where it was previously more difficult to derive rules, as human logic can be used to aid in reward function creation, as shown in the example for Shovel Knight. This method also leads itself to be explored by explainable approaches in determining the reason for decision making.

Acknowledgment

I would like to thank Dr. Barnabas Bede, Dr. Yilin Wu, and Shivam Kumar for all the help they have given me for this project. I would also like to thank DigiPen Institute of Technology as well as Prof. Peter Toth, Dr. Ola Amayri, Dr. Brigitta Vermesi, and Dr. Pushpak Karnick who all helped me realize my love for machine learning and math. And one last thanks to Eric Zander for collaborating on other projects and papers this past year.

References

[1] D. Malik, Y. Li, and P. Ravikumar, "When is generalizable reinforcement learning tractable?" *CoRR*, vol. abs/2101.00300, 2021. [Online]. Available: https://arxiv.org/abs/2101.00300

[2] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[3] E. Hernandez, S. Schwettmann, D. Bau, T. Bagashvili, A. Torralba, and J. Andreas, "Natural language descriptions of deep visual features," *CoRR*, vol. abs/2201.11114, 2022. [Online]. Available: https://arxiv.org/abs/2201.11114

[4] "Shovel knight: Shovel of hope." [Online]. Available: https://www.yachtclubgames.com/games/shovel-knight-shovel-of-hope

[5] "Super mario bros.: The lost levels," 1986.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[7] C. Kauten, "Super Mario Bros for OpenAI Gym," GitHub, 2018. [Online]. Available: https://github.com/Kautenja/gym-super-mario-bros

[8] P. Skalski, "Make Sense," https://github.com/SkalskiP/make-sense/, 2019.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *CoRR*, vol. abs/1406.4729, 2014. [Online]. Available: http://arxiv.org/abs/1406.4729

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[11] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[12] G. Jocher, "YOLOv5 by Ultralytics," 5 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[13] C. Wang, I. Yeh, and H. M. Liao, "You only learn one representation: Unified network for multiple tasks," *CoRR*, vol. abs/2105.04206, 2021. [Online]. Available: https://arxiv.org/abs/2105.04206

[14] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[15] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[16] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[17] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[18] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/11796

[19] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, Mar. 2016. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/10295

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, cite arxiv:1511.05952Comment: Published at ICLR 2016. [Online]. Available: http://arxiv.org/abs/1511.05952

[21] Z. Wang, N. de Freitas, and M. Lanctot, "Dueling network architectures for deep reinforcement learning," *CoRR*, vol. abs/1511.06581, 2015. [Online]. Available: http://arxiv.org/abs/1511.06581

[22] K. D. Asis, J. F. Hernandez-Garcia, G. Z. Holland, and R. S. Sutton, "Multi-step reinforcement learning: A unifying algorithm," *CoRR*, vol. abs/1703.01327, 2017. [Online]. Available: http://arxiv.org/abs/1703.01327

[23] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," *CoRR*, vol. abs/1707.06887, 2017. [Online]. Available: http://arxiv.org/abs/1707.06887

[24] M. Fortunato, M. G. Azar, B. Piot, J. Menick, M. Hessel, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy networks for exploration," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=rywHCPkAW

[25] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[27] Yannbouteiller, "vgamepad: Virtual xbox360 and dualshock4 gamepads in python." [Online]. Available: https://github.com/yannbouteiller/vgamepad

[28] B. Höglinger-Stelzer, "Vigem: Virtual gamepad emulation framework." [Online]. Available: https://vigem.org/

[29] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755.

[30] "Mss: An ultra fast cross-platform multiple screenshots module in pure python using ctypes." [Online]. Available: https://pypi.org/project/mss/

[31] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[32] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6469–6477.

[33] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan, "Learning to utilize shaping rewards: A new approach of reward shaping," *CoRR*, vol. abs/2011.02669, 2020. [Online]. Available: https://arxiv.org/abs/2011.02669