Extended Abstract

Motivation Deep Reinforcement Learning (DRL) has achieved notable success in games ranging from Atari to strategic board games like Go and Chess. However, fewer studies have explored their performance in complex, visually-rich, real-time action games such as *Hollow Knight*. These environments present unique challenges: high-dimensional pixel-based state representations, intricate real-time action spaces demanding precise timing, significant resource constraints due to real-time decision making, and sparse, delayed rewards. Our work investigates whether DRL algorithms can succeed in such resource-constrained, visually intensive scenarios.

Method We developed a computationally efficient RL framework to train agents capable of defeating bosses in *Hollow Knight*, starting with the Gruz Mother fight. Observations are grayscale, downsampled (192×192 pixels) screenshots stacked in groups of four frames to capture temporal dynamics. The action space was simplified to discrete combinations of movements, attacks, and displacements, significantly constraining the game's inherently complex mechanics. We experimented with multiple DRL algorithms: standard DQN, Double DQN, Dueling DQN, Double Dueling DQN (D3QN), and Discrete SAC, comparing their effectiveness in the described environment.

Implementation Observations were processed through a pretrained SqueezeNet CNN architecture to produce compact (512-dimensional) feature vectors. Value-based approaches (DQN variants) used multilayer perceptrons (MLP) with two layers (512 units each) to predict Q-values. Double DQN introduced separate online and target networks to reduce overestimation bias, while Dueling DQN decomposed Q-values into separate state-value and advantage streams. Discrete SAC, our policy-based method, utilized separate actor and critic networks trained via entropy-regularized actor-critic methods, employing differentiable policy sampling through Gumbel-Softmax. Agents were trained via episodes interleaved with gradient updates from a replay buffer populated initially by random exploration data and then by policy rollouts.

Results Our experiments demonstrated that the simple DQN consistently outperformed more theoretically sophisticated methods, achieving the highest average evaluation reward (4.393) and low variance. Dueling DQN (3.862) and Double DQN (3.563) were slightly less effective, while Double Dueling DQN (2.281) exhibited instability. Discrete SAC performed worse than the random baseline, highlighting difficulties with policy stability and complexity in this environment. Qualitatively, successful models converged to trivial, repetitive strategies (e.g., repeatedly jumping and attacking in place), whereas complex methods displayed erratic or ineffective behaviors.

Discussion The limited expressivity of our chosen architectures, constrained by the computational resources of a CPU-only environment, led agents toward simple, repetitive policies rather than nuanced, context-sensitive strategies. The failure of more sophisticated algorithms indicates potential sensitivity to hyperparameters. Future experiments employing GPUs or other hardware acceleration could allow deeper, more expressive models, potentially enabling more sophisticated, adaptive policies. Additionally, data augmentation or advanced regularization techniques might address stability issues and enhance policy diversity.

Conclusion We demonstrate that straightforward value-based methods, specifically the DQN, effectively learn stable but simple strategies in visually dense, resource-constrained real-time environments. Despite theoretical appeal, more complex algorithms, particularly Discrete SAC and Double Dueling DQN, struggled significantly in practice.

A Truly Hollow Knight: Resource Constrained Visually Dense Boss Fights

Anthony Maltsev
Department of Computer Science
Stanford University
amaltsev@stanford.edu

Abstract

In this report, we present a CPU-only, real-time deep-reinforcement-learning agent trained to fight bosses in Hollow Knight using only visually dense observations. We use an OpenAI gym interface wrapper to create an interactive real-time environment for the game. We compare five discrete-action RL algorithms: DQN, Dueling DQN, Double DQN, Double Dueling DQN, and Discrete SAC. After training models to convergence, we find that a straightforward DQN achieves the best results, demonstrating that a straightforward DQN architecture can outperform more complex extensions in a visually dense, constrained environment.

1 Introduction

Deep Reinforcement Learning (DRL) methods have demonstrated impressive capabilities across a variety of gaming environments, ranging from classic Atari games Mnih et al. (2013) to strategic board games such as Go and Chess Silver et al. (2016, 2017). However, less attention has been given to their application in more complex, visually dense, and mechanically demanding real-time games. This project investigates the application of RL to boss fights in *Hollow Knight*, a popular 2D action-platformer known for its challenging and precise combat mechanics.

The task of defeating boss characters in *Hollow Knight* presents a challenging environment for reinforcement learning due to several factors. First, the game's state is high dimensional and visually rich, composed of continuous streams of pixel-based observations. Second, the action space, although discrete, involves intricate timing and combinations of movements, attacks, and displacement operations (eg dashing, different height jumps). Third, the real-time nature of the game requires decisions to be computed rapidly, within fractions of a second (approximately 0.166 s per step), imposing resource constraints on model inference time. Lastly, the reward structure is naturally sparse, typically awarded only upon the successful completion of the battle, necessitating thoughtful reward shaping for effective learning.

To address these challenges, this project has three primary objectives:

1. To develop a computationally efficient DRL agent capable of real-time inference under significant hardware constraints (CPU-only training and inference).
2. To systematically benchmark and compare multiple well-established DRL algorithms, Deep Q-Networks (DQN) Mnih et al. (2013), Dueling DQN Wang et al. (2016), Double DQN van Hasselt et al. (2015), and Soft Actor-Critic (SAC) adapted for discrete action spaces Christodoulou (2019), on the visually complex setting of boss battles in *Hollow Knight*.
3. To assess robustness of these DRL methods when trained on visually dense observations, evaluating their ability to learn meaningful combat strategies against the Gruz Mother boss, one of the more simple bosses in the game.

We implement our experimentation using a customized *Hollow Knight* environment wrapped in an OpenAI Gym interface based on prior work Yang (2023). Observations consist of downsampled, grayscale screenshots captured from gameplay, and each state is represented by the four most recent frames to provide some minimal temporal context. Since the environment steps at a regular interval of 0.166s, these four frames represent two thirds of a second of temporal context, enough to provide context about the boss's current movement.

Our experimental results reveal some insights into the suitability of various RL methods for resource-constrained, visually intensive real-time environments. Specifically, we find that DQN-based methods substantially outperform more complex alternatives such as Discrete SAC, which failed to progress beyond random-level performance, and Double DQN, which degraded performance without providing training stability.

This report documents the design and construction of the environment, the formulation of reward functions, the selection and training of DRL algorithms, and detailed analyses of their performance. We conclude by discussing broader implications for deploying DRL algorithms in similar gaming contexts and suggest potential future research directions, such as exploring additional data augmentation and training stabilization techniques to further improve agent robustness and generalization capabilities.

2 Related Work

Deep reinforcement learning has achieved significant success in gaming domains, particularly with Deep Q-Networks (DQNs), which leverage convolutional neural networks to enable agents to learn directly from raw pixel data Mnih et al. (2013). The original DQN framework introduced experience replay and target networks to stabilize training, addressing issues such as correlated samples and non-stationary targets. To further enhance stability and accuracy of Q-value estimates, subsequent variants of DQN were developed.

Double DQN van Hasselt et al. (2015) addresses the overestimation bias present in standard DQNs, which arises because the same network is used for action selection and value estimation. By maintaining two separate networks—one for selecting the optimal next action and another (target network) for evaluating its value—Double DQN reduces the tendency to overestimate Q-values, thereby aiming for more accurate and stable training. In contrast, Dueling DQN Wang et al. (2016) introduces a structural modification that decomposes the Q-value into two separate streams: one estimating the value of a state (independent of the action) and another estimating the advantage of each action. This decomposition enables more efficient learning by helping the network distinguish between actions that meaningfully differ in their impact versus those that are less influential, especially useful in scenarios where certain states have uniform values across actions.

Beyond DQN-based methods, policy-gradient approaches like Proximal Policy Optimization (PPO) Schulman et al. (2017) and Soft Actor-Critic (SAC) Haarnoja et al. (2018) have been explored to improve exploration and policy robustness. SAC, specifically, uses entropy maximization to encourage exploration and improve sample efficiency in environments where effective exploration can be challenging. Additionally, recent model-based approaches like OC-STORM Zhang et al. (2025) explicitly model environment dynamics to plan actions more efficiently, although such methods typically require greater computational resources and complexity compared to their model-free counterparts.

3 Environment

The environment, based on work in Yang (2023), captures state information directly through visual observations, taking screenshots of the game rendered at each timestep. To keep the problem tractable and reduce computational demands, the raw observations are first downsampled to 192×192 pixels and converted to grayscale, reducing the momory footprint by 75x. Four consecutive frames are stacked together at each step, allowing the agent to perceive temporal dynamics crucial for reacting to enemy animations, movements, and attacks. Figure 1 provides an illustration of this pipeline, demonstrating the transition from the raw visual inputs to the processed grayscale images that serve as input to the feature extractor component of the agent.

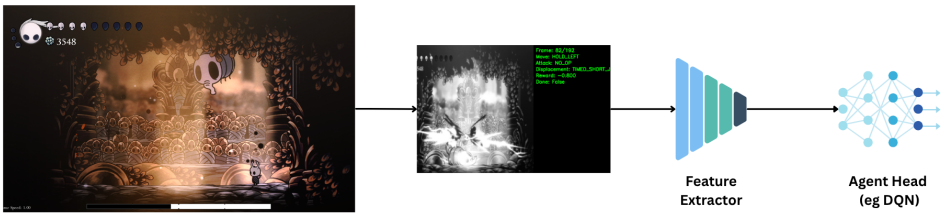

Figure 1: Environment Diagram. Observations are downsampled, then fed to a pretrained feature extractor, then fed into an action prediction head, such as a DQN.

We define the agent's action space as a discrete cartesian product of three separate dimensions: movement (left, right, no action), attack (attack, no action), and displacement (short jump, no action). This action formulation simplifies the problem compared to the full action set while still providing enough expressive power to perform effectively against the game's bosses. The base game has significantly more actions, such as longer jumps, timed healing actions, spells, dashes, charged attacks, and so on. We restrict the action set to a small subset so that the output space is not too large and the task is more feasible. Actions are executed at approximately 0.166-second intervals, requiring agents to make decisions quickly and efficiently.

The environment includes explicit reward shaping to encourage effective combat behaviors. The reward function assigns positive rewards for dealing damage to bosses and negative rewards for receiving damage or idling without interaction. This approach ensures intermediate feedback during training, mitigating the sparse-reward challenge typically associated with defeating bosses, which otherwise might severely hinder the RL agent's ability to learn effective strategies. In order to facilitate these intermediate rewards, we use a game modification file, freely distributed through Lumafly et al (2023) which adds a visible boss health bar to the game. This health bar is recognized by an observation script during the environment step which allows the agent to track the player and boss health to facilitate this reward calculation.

Event	**Reward**
Player Hurt	-0.8
Enemy Hurt	0.5
No Event	-0.01

Table 1: Reward function for the environment

We initially ran tests without the small negative reward for idling in the environment but we observed that the agent would prioritize staying in a corner, which reduced the chance of being hit, but would also make the rollouts very long.

4 Methods

Our framework consists of three primary components: feature extraction from observations, reinforcement learning models, and training procedures.

4.1 Feature Extractors

The first component is a feature extractor, which processes frames from the environment into a much lower-dimensional feature representation which the RL model heads can use. We experimented initially with custom convolutional neural networks (CNNs) of varying sizes, but they were not able to learn good features quickly enough during training and we observed that the model would get stuck performing some useless action, such as running into one of the walls. We then experimented with different pretrained models, exploring different models with pretrained weights available from torchvision. We first experimented with MobileNet **?** architectures, because they are famousaly lightweight with good results on ImageNet. However, even the smallest MobileNet model would exceed the 166ms inference time constraint set by the environment step time. We settled upon using a pretrained SqueezeNet architecture Iandola et al. (2016), due to its lightweight footprint (roughly 5 MB) and computational efficiency. The smallest SqueezeNet model would run feature extraction in about 80ms on our hardware, sufficient for real time inference.

The output of the feature extractor would be a 512 dimensional vector. This vector forms the representation for downstream reinforcement learning models.

4.2 Reinforcement Learning Models

We explored both value-based and policy-based reinforcement learning approaches, implementing variants of Deep Q-Networks (DQNs) and a discrete Soft Actor-Critic (SAC) architecture.

4.2.1 Value-Based Models:

Our value-based approaches included a standard DQN Mnih et al. (2013), Double DQN van Hasselt et al. (2015), Dueling DQN Wang et al. (2016), and Double Dueling DQN (D3QN) Cheng et al. (2022). Each of these models share the 512-dimensional feature vector from the extractor layer as input.

DQN: The basic DQN employs a straightforward multilayer perceptron (MLP) comprising two fully connected layers, mapping from the 512-dimensional feature representation to 512 hidden units, then to Q-values for the 12 discrete actions in our environment.

Double DQN: To mitigate the well-known overestimation bias of standard DQN, the Double DQN uses two networks: an online network selects the optimal next action, while a separate, periodically updated target network evaluates its corresponding Q-value. Architecturally, it mirrors the DQN model structure.

Dueling DQN: This variant separates the final fully-connected layers into two streams: a value-stream, outputting a scalar state-value, and an advantage-stream, predicting the advantage values of each action. Both streams contain two-layer MLPs with 512 hidden units each. The final action-value is calculated as the sum of state-value and mean-normalized advantages.

Double Dueling DQN (D3QN): This variant combines both the Dueling and Double DQN approaches. It uses the dueling architecture to separately estimate the state-value and advantages for each action, and integrates the Double DQN update strategy to reduce overestimation. Specifically, the online network's advantage stream is used to select the best next action (via $\arg\max$), while the target network computes the Q-value associated with that action. This is meant to result in both more stable training and better value estimation in environments with noisy or redundant actions.

4.2.2 Policy-Based Models:

We also implemented a policy-based approach—Discrete Soft Actor-Critic (Discrete SAC) Christodoulou (2019), a version of SAc naturally adapted for discrete action spaces. Our discrete SAC model consists of separate actor and critic networks. The actor network is a two-layer MLP (512 units each) predicting logits for a softmax distribution over discrete actions. Differentiable policy sampling is implemented using the Gumbel-Softmax trick. Two separate critic networks estimate action-values (Q-values) independently to mitigate overestimation. Each critic comprises two fully-connected layers, receiving as input the concatenation of the 512-dimensional feature vector and a one-hot encoding of the selected action, mapping to a single scalar Q-value prediction.

The discrete SAC algorithm optimizes both expected return and entropy-regularization objectives, balancing exploration and exploitation. A learned temperature parameter dynamically controls the exploration-exploitation trade-off, optimizing towards a target entropy to maintain robust exploration.

We chose to explore policy based methods after observing that all of the value-based methods would usually choose a single action which yields good results, instead of using varied actions depending on the state environment. Since the actions are sampled from stochastic policy, it is more likely that the agent will employ a variety of different actions in its strategy.

4.3 Trainers

We implemented two distinct training procedures tailored to the value-based and policy-based models, respectively.

Value-Based Trainer: Our value-based trainer implements standard Q-learning training loops. Episodes begin with environment resets and proceed stepwise. The model is updated after each rolout using batches of experiences from a replay buffer which stores the explorations of a random policy and the rollouts from the current and past agents. A temporal difference (TD) loss is computed between current Q-value predictions and target Q-values generated from reward signals and the subsequent state's Q-values predicted by a periodically updated target network. Gradients from the TD loss are backpropagated through the network, and gradient clipping to a magnitude of 10 is employed for numerical stability. This is a standard training procedure for DQNs.

Policy-Based Trainer (Discrete SAC): The policy-based trainer, tailored for discrete SAC, implements a more complex actor-critic training loop. Each training step involves multiple gradient updates for the actor network, critic networks, and the entropy temperature parameter, α. Critic updates minimize the TD-error between the predicted and target Q-values, while the actor network is trained to maximize expected returns and entropy via soft policy updates. Temperature parameters are adjusted to maintain a predefined target entropy, encouraging exploration. Target critic networks are updated using soft updates, enhancing stability in Q-value estimates during training. We use a γ value of 0.99 for entropy regularization and a τ value of 0.005 for target updates.

5 Experimental Setup

We precomputed a pool of roughly 11000 observation-action-reward tuples by observing 75 rollouts of a random policy, which would choose any of the 12 actions uniformly at random at each timestep, to populate the replay buffer. This replay buffer would be used for each of the methods we described above to perform initial training.

For each of the 5 methods described above, we trained an agent using the following experimental setup. The model would be initialized and allowed to train on the precomputed replay buffer of random actions for 50 32-sized batches of experiences from the buffer. Then, the model would alternate between collecting a policy rollout and training about 300 times (roughly 8 hours). We found that the models would roughly converge after this amount of time.

We look at the metric of reward per rollout as the primary metric for policy evaluation.

6 Results

6.1 Quantitative Evaluation

The quantitative results in Table 2, along with the corresponding training curves shown in Figure 2, highlight some of the differences in performance and stability across the value-based and policy-based reinforcement learning algorithms tested. Among all methods, the standard DQN achieved the highest average evaluation reward, alongside relatively low variance, indicating both strong performance and consistent behavior. Dueling DQN and Double DQN followed closely, with average rewards of 3.862 and 3.563 respectively, suggesting that both architectural and algorithmic modifications to vanilla DQN provide modest but measurable improvements in value estimation and training dynamics.

Algorithm	Avg. Eval Reward	Std. Dev.
Random (Baseline)	-0.247	1.621
DQN	4.393	1.687
Dueling DQN	3.862	1.932
Double DQN	3.563	1.863
D3QN	2.281	4.347
Discrete SAC	-3.094	1.780

Table 2: Evaluation average reward and standard deviation for each algorithm.

Interestingly, Double Dueling DQN (D3QN) underperformed compared to the other DQN variants, with a significantly lower average reward and a large standard deviation. This indicates that the training was unstable and the learned policy produced inconsistent behavior. We see in the black trace in the learning curve that there were some rollouts in which the model catastrophically failed, yielding reward on the order of -10, corresponding to dying having dealt no damage to the boss. While in theory D3QN combines the advantages of both Dueling and Double DQN, we observe that it has significantly worse performance in practice. We hypothesize that it may suffer from compounded sensitivity to hyperparameter assignment as there are more knobs to be tuned here. We mostly copied learning rate hyperparameters that worked for the DQN, hoping they would generalize to this model as well.

Finally, Discrete SAC showed poor performance, achieving an average reward of worse than even the random baseline. Despite its theoretical advantages in exploration since it uses a stochastic policy, we observe that this model struggled to learn any meaningful policy in this setting. This is reflected in both its low reward and flat learning curve (purple). This result is likely due to the added complexity of actor-critic training which trains several different models at the same time. Without a very strong signal, this multi model training might interfere with the other models. Additionally, we used the same learning rate as for DQN, which might not generalize well to this architecture.

These findings collectively suggest that for this task, simpler value-based methods (particularly the standard DQN) can outperform more sophisticated alternatives in both performance and stability.

6.2 Qualitative Analysis

Qualitatively, we observe that the 3 top performing models (DQN, Double DQN, Dueling DQN) all converged to the same strategy. The agent would repeatedly jump up and down in place, attacking by swinging their sword. When the boss would move behind the player character, the agent would turn around and swing the other way, but other than that, the model honed in on repeating this one action. This happened consistently across three different model architectures. A screenshot of this is provided in Figure 3.

The D3QN model converged to a different strategy. This agent would run into the wall repeatedly repeatedly jumping, attacking and returning to the wall. This allowed the model to attack and turn around almost instantly, an exploit of the game engine that would not otherwise be possible given our models are restricted to taking actions at regularly spaced intervals. Again, like the other models, this model repeatedly took one action without a lot of variability.

The discrete SAC model took a lot more varied actions, moving around the screen, jumping and attacking in a manner more similar to the random model. We observe that many times the player character controlled by this agent would jump towards the boss to try and attack it, but this would cause it to take damage due to collision with the boss. We believe that this aggressive behavior caused the model to fail, compared even to the baseline random model. With a more expressive model architecture (such as a better feature extractor and deeper agent head models) the agent would be able to learn more fine-grained movement controls and succeed more.

As a rough baseline, a reward of > 1.5 would usually correspond to a victory against the boss, and less than that would usually correspond to a loss.

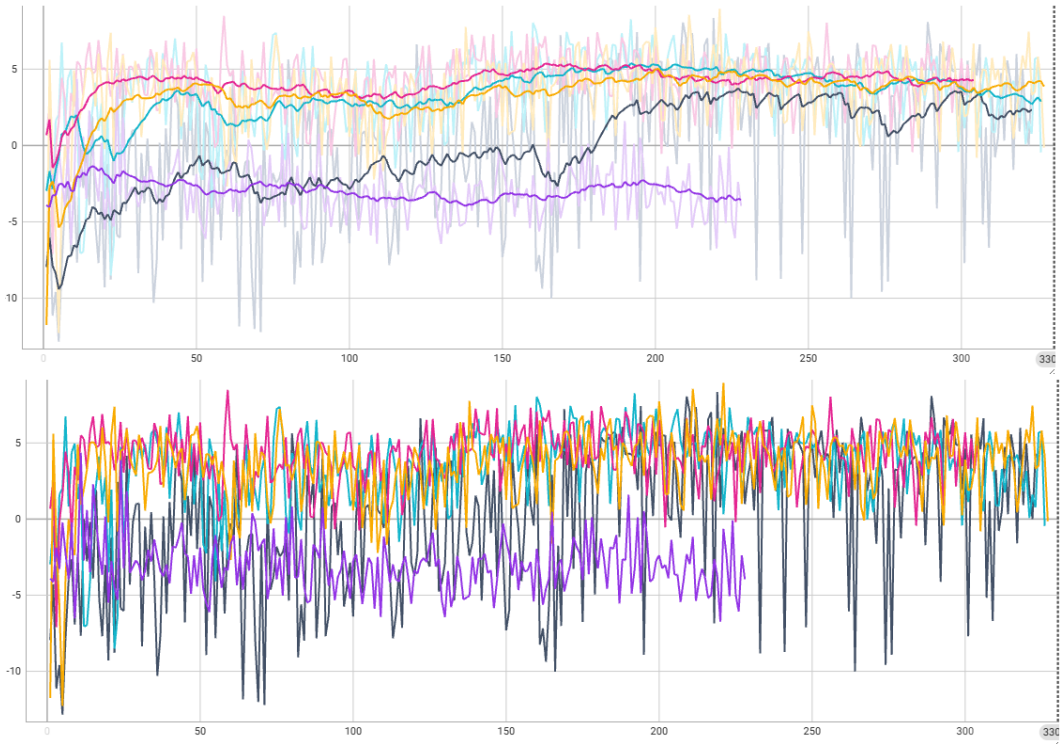

Figure 2: All policy return over training time curves. Top is smoothed, Bottom is raw. DQN = Pink, Double DQN = Blue, Dueling DQN = Yellow, D3QN = Black, Discrete SAC = Purple

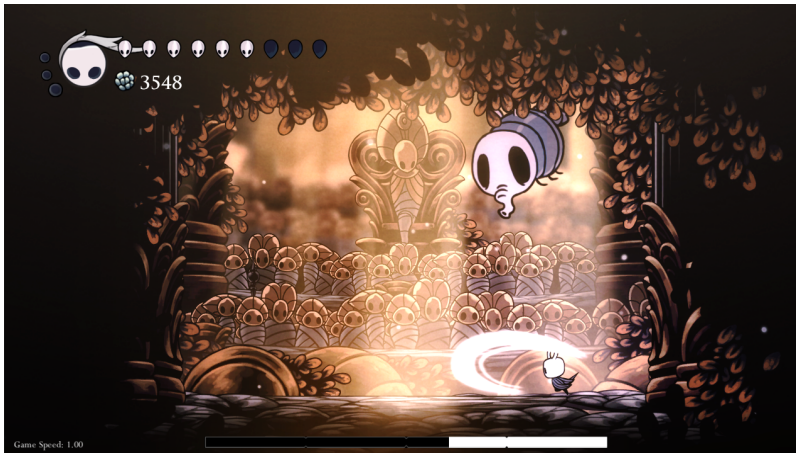

Figure 3: DQN Strategy of just jumping and attacking in place. The player character (bottom right) jumps up and down attacking (the white arc in front of the character is the attack) and waits for the boss to come to it

7 Discussion

In general, we observe that given the limited expressivity of the model architectures that we used, the models were not able to learn sophisticated strategies that would be variable depending on the state, instead favoring strategies that would take consistent actions that would yield decent results regardless of the observed state. The simple DQN model which was able to hone in on a good action to repeat almost invariably performed the best.

An interesting experiment would be to use a machine with a GPU or other hardware accelerator attached which would enable more expressive models to be used. We believe that this would improve the model performance and allow for more sophisticated strategies to be learned.

8 Conclusion

In this work, we developed and evaluated several deep reinforcement learning algorithms in the context of resource-constrained, visually dense, real-time gameplay environments, specifically focusing on boss battles in the game *Hollow Knight*. Our results show that relatively simple, value-based methods—particularly the standard DQN—consistently outperformed more theoretically sophisticated alternatives such as Discrete SAC and Double Dueling DQN. These simpler models converged to stable, albeit somewhat trivial, policies characterized by repetitive actions optimized for consistency over varied gameplay states.

These findings suggest significant room for future work, especially in exploring computationally feasible model enhancements, data augmentation techniques such as SVEA, and regularization strategies capable of improving generalization and encouraging diverse strategies. Utilizing hardware accelerators to accommodate deeper architectures could facilitate learning more state-dependent policies.

9 Team Contributions

- **Anthony Maltsev:** everything

Code for this project can be found at:

`https://github.com/anthonymaltsev/HKRL`

Changes from Proposal We initially planned to initialize the model with behavioral cloning based on rollouts of humans fighting the boss. We tried modifying the environment to capture human inputs, but they are too far out of distribution since they do not happen at regular intervals (the autonomous agent is set up to take actions at regularly space $\frac{1}{6}$ second intervals). Also, the humans are able to do multiple actions sequentially in a short amount of time such as attacking and instantly turning around to run away. The human expert examples that we were able to capture were too far out of distribution for the agent to be able to learn from. Instead, we relied on an exploration phase in which a random policy was used to collect offline experience for the models to learn from.

References

Weijun Cheng, Xiaoshi Liu, Xiaoting Wang, and Gaofeng Nie. 2022. Task Offloading and Resource Allocation for Industrial Internet of Things: A Double-Dueling Deep Q-Network Approach. *IEEE Access* 10 (2022), 103111–103120. `https://doi.org/10.1109/ACCESS.2022.3210248`

Petros Christodoulou. 2019. Soft Actor-Critic for Discrete Action Settings. arXiv:1910.07207 [cs.LG] `https://arxiv.org/abs/1910.07207`

Hussain Aziz et al. 2023. Lumafly: Hollow Knight Mod Platform. `https://github.com/TheMulhima/Lumafly`

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. arXiv:1801.01290 [cs.LG] `https://arxiv.org/abs/1801.01290`

Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. arXiv:1602.07360 [cs.CV] `https://arxiv.org/abs/1602.07360`

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602 [cs.LG] `https://arxiv.org/abs/1312.5602`

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] https://arxiv.org/abs/1707.06347

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv preprint arXiv:1712.01815* (2017).

Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep Reinforcement Learning with Double Q-learning. arXiv:1509.06461 [cs.LG] https://arxiv.org/abs/1509.06461

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. Dueling Network Architectures for Deep Reinforcement Learning. arXiv:1511.06581 [cs.LG] https://arxiv.org/abs/1511.06581

Zhantao Yang. 2023. HKRL, https://github.com/seermer/HollowKnight_RL. https://github.com/seermer/HollowKnight_RL

Weipu Zhang, Adam Jelley, Trevor McInroe, and Amos Storkey. 2025. Objects matter: object-centric world models improve reinforcement learning in visually complex environments. arXiv:2501.16443 [cs.LG] https://arxiv.org/abs/2501.16443