Gaming the Video Against Yourself

Motivation Real-world decision-making is inherently complex and partially observable, requiring agents to interpret high-dimensional sensory input and take reward-maximizing actions with incomplete information. This challenge is mirrored in adversarial video game settings, where agents must infer game state from raw visual input and react to dynamic, opposing players. The goal of this work is to explore whether an autonomous agent can learn spatially and temporally aware behaviors from raw pixel-input in a two-player toy game.

Method To address the challenges of learning in a visually complex, partially observable, multi-agent setting, this work combines convolutional neural networks (CNNs), long short-term memory networks (LSTMs), and the Soft Actor-Critic (SAC) algorithm, as well as periodic updates to the enemy policy for self-play. The agent processes sequences of raw RGB frames to form latent feature representations. These are used by an actor to select actions and by a critics to estimate expected returns. The policy is trained using self-play and a replay replay buffer, with entropy regularization to limit excessive exploration.

Implementation A custom two-player game was built using pygame. The game contained resource bars, ability cooldowns, and a blank playing space. The agent receives sequences of visual input at a resolution of 200×150 pixels. Three separate CNN-LSTM pipelines output features to the actor and critic networks. The actor returns a discrete action type and continuous-valued position, while the critic predicts a scalar eestimated return value for each transition. The action space was gradually simplified during training to reduce instability, eventually constraining one agent to one action while the enemy remained passive. Full stamina regeneration was also introduced to prevent resource constraints from dominating the learning process.

Results The trained agent consistently outperformed a random baseline, achieving average rewards of 295 to the random baseline's 44. Additionally, the trained agent had a quicker time-to-kill, though the random policy was unable to kill within the maximum episode length. Despite significant actor and critic losses, the agent learned an effective policy. Visualizations showed the trained policy firing many projectiles towards the enemy. Loss plots revealed growing critic loss and decreasing actor loss, indicating difficulty in value estimation and policy optimization, likely due to instability from the intertwined learning objectives.

Discussion The unexpected success of the agent despite poor losses suggests that the simplicity of the environment may have allowed for high returns through suboptimal strategies. The simple environment might have also made it easier for the agent to stumble into a reward-maximizing policy by accident. Curriculum learning approaches, such as progressively expanding the action space or increasing opponent complexity, could encourage more generalizable behavior.

Conclusion This work shows that even with limited supervision and raw visual input, agents can learn realistic behaviors using CNN-LSTM models and Actor-Critic methods in a self-play setting. Although training was unstable and required environment simplification, the agent achieved strong performance. Future improvements could target reward shaping, curriculum design, and alternative baselines to better align learned behavior with desired outcomes and to appropriately evaluate the strength of a policy.

Gaming the Video Against Yourself

Joshua Boisvert
Department of Aeronautics and Astronautics
Stanford University
joshmboi@stanford.edu

Abstract

This paper investigates the integration of computer vision, deep reinforcement learning, and self-play for training an autonomous agent to play a custom two-player game inspired by League of Legends and Dota 2. The environment, built in pygame, features discrete abilities, resource management, and spatial positioning. To process visual input and capture temporal dependencies, the agent employs a convolutional neural network (CNN) combined with a long short-term memory (LSTM) module. Policy learning is conducted using a Soft Actor-Critic (SAC) framework enhanced with continuous action embeddings and entropy regularization, leveraging a replay buffer and self-play dynamics. Experimental results show that, although the agent reliably exploits the environment to maximize rewards, the actor and critic fail to model the environment dynamics accurately. These findings underscore the challenges of learning in partially observable, multi-agent settings and point to potential improvements through curriculum learning and architectural refinement.

1 Introduction

The real world presents a dynamic and complex environment, governed by physical laws that are often only approximated through simplified theorems and models. Despite these abstractions, extracting meaningful information from sensory inputs remains an inherently difficult task. For artificial agents to operate effectively in such settings, they must first perceive and interpret their surroundings and then translate those perceptions into purposeful actions.

Mapping raw sensory data to a meaningful internal representation or "state" is nontrivial, particularly in high-dimensional, partially observable environments. Moreover, to handle diverse tasks, agents must generalize across broader feature spaces rather than rely on hand-crafted, task-specific inputs. This requires that the agent learn to focus on relevant features through experience, rather than human-defined heuristics.

Action selection is similarly challenging. Feedback is often sparse, delayed, or ambiguous, and the definition of "success" can be fluid or context-dependent. For example, while one could heuristically reward a walking agent based on time spent airborne, this fails to capture the nuanced coordination of leg movement, potentially encouraging an agent to stand still. Furthermore, such feedback must be derived from raw input and without relying on pre-processed signals. In reinforcement learning, this complexity is distilled into the design of reward functions, but imperfect or sparse rewards make it difficult for agents to learn effective behaviors. Mapping high-dimensional sensory observations to coherent, goal-directed actions remains a fundamental challenge.

Video games present a simplified yet realistic model for these challenges. They provide raw visual and auditory input and require action sequences to manipulate game state. Players must infer game conditions (e.g., health, position, objectives) from pixels alone and make decisions under

uncertainty and adversarial pressure. These properties make games an incredible space for exploring perception-driven decision-making.

This paper investigates whether an artificial agent can learn to map visual input to reward-maximizing actions in a custom two-player game. The environment, built using pygame, features abilities with spatial effects, resource management, and adversarial dynamics. The agent receives sequences of 12 RGB frames (200×150 resolution) and processes them using a convolutional neural network (CNN) followed by a long short-term memory (LSTM) network. While training, three parallel CNN-LSTM blocks are used, with two for two critics and one feeding into an actor network. The critic networks estimate expected reward, while the actor network selects actions. The critic is a fully connected network (FCN) producing a scalar value, while the actor is an FCN with two output heads: one for discrete action selection and another for continuous positional targeting.

2 Related Work

Early work in deep reinforcement learning (RL) has demonstrated that agents can learn to act from raw visual input. Mnih et al. (2013) introduced a method where agents use stacks of four frames as input, enabling them to capture temporal dynamics such as motion and momentum. This frame-stacking approach works well in fully observable settings, but is less effective in partially observable environments like the one in this paper, since agents lack access to opponents' hidden states or internal decisions. As a result, the agents rely on recurrent architectures to capture temporal dependencies.

In contrast, Lange et al. (2012) focused on learning latent representations of raw pixel data through convolutional encoders, providing a compressed feature space for policy learning. This approach improves both sample efficiency and generalization while simplifying downstream policy models. Inspired by this, I used a similar architecture to encode the visual state into a latent space before action selection.

Another influential example comes from Lample and Chaplot (2018), who trained an agent to play DOOM using a CNN-LSTM pipeline. Here, convolutional layers extract spatial features from raw frames, while the LSTM maintains a temporal hidden state. While effective in capturing dynamics in a single-agent setting, this architecture lacks the complexity needed in adversarial environments. Without opponent modeling or self-play, the agents struggle to generalize beyond fixed, predictable opponents. It is worth noting that DOOM itself has a fair amount of stochasticity. Even so, the environments are fairly consistent through different playthroughs.

To address this limitation, multiple works (Dwibedi and Vemula (2020), Bansal et al. (2018), OpenAI et al. (2019)) incorporate self-play as a means of generating a curriculum and promoting robustness. In particular, OpenAI et al. (2019) introduced using a pool of past policies to avoid overfitting and instability when training. Self-play introduces stochasticity and ensures continuous adaptation, making it well-suited for multi-agent environments.

For the underlying RL algorithm, this work builds off the Soft Actor-Critic (SAC) framework introduced by Haarnoja et al. (2019). SAC combines off-policy learning with entropy regularization to encourage exploration and stabilize updates. Its replay buffer and off-policy formulation make it effective in symmetric, self-play settings where the enemy's policy is updated with the player's policy. Further extensions like Kim et al. (2020) adapt SAC to vision-based tasks by incorporating convolutional encoders that map raw image input into latent states. These models do away with traditional Q-learning in favor of Actor-Critic methods, offering better stability in continuous or partially observable domains.

These prior efforts inform the design choices in this paper. By combining convolutional encoders, recurrent networks, self-play, and the SAC algorithm, I aim to demonstrate that an agent can learn capable strategies in visually rich, adversarial, two-player settings.

3 Methods

To support my agent's learning, I developed a custom two-player video game and paired it with a deep learning system built around CNN-LSTM modules for state representation and Actor-Critic networks for decision-making and value estimation.

3.1 Game Environment

The environment is a two-player MOBA-inspired game developed using the pygame library, drawing design elements from titles like League of Legends and Dota 2. Core mechanics of the game include movement, ability usage, and resource management. Each player can perform one of three abilities. The player can fire a projectile ("Q"), plac a damage zone ("W"), and use a shield ("E"). These abilities have varying cooldowns and effects, resulting in the ability for strategic diversity.

The mouse position is used for directional input, introducing spatial reasoning challenges. Additionally, To reduce computational demands for the learning agent, I increased sprite sizes and downscaled the visual resolution from 800×600 to 200×150. Health and stamina bars are rendered for both players on separate screens. Figure 1a shows an example screenshot of agents playing using random policies.

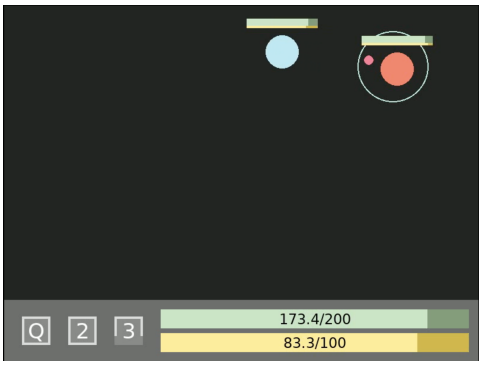
3.1.1 CNN-LSTM Architecture

Visual input is processed through a convolutional neural network followed by a long short-term memory (LSTM) network. The CNN extracts spatial features, and the LSTM captures temporal dependencies across frames, which is important due to the partial observability of internal states like cooldowns.

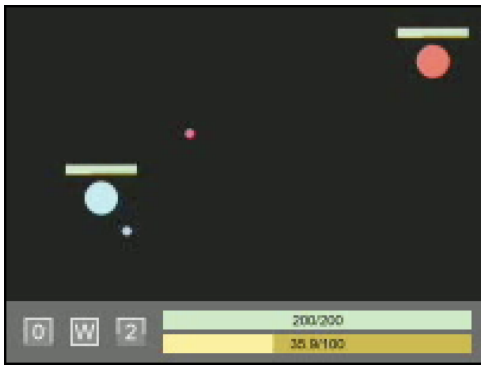
The CNN processes a 12-frame sequence of $200 \times 150 \times 3$ RGB images using three convolutional layers.

- Conv1: 16 filters, kernel size 3, stride 2, padding 1
- Conv2: 32 filters, kernel size 3, stride 2, padding 1
- Conv3: 32 filters, kernel size 3, stride 2, padding 1

Each convolutional layer is followed by a ReLU activation and a 2×2 max pooling layer. The final output is flattened and projected to a 64-dimensional feature vector. Figure 1b shows a sample reduced-resolution input.



(a) Example image of toy game environment.



(b) Example input to CNN.

This feature vector is passed into an LSTM with a hidden size of 128 for sufficient ability to fit the data. The LSTM maintains a temporal context and hidden state, enabling the agent to reason about sequential dependencies in the environment.

The rationale for selecting a CNN-LSTM setup for translating the raw input into a latent feature space lies in being able to process visual input with placing high emphasis on locality, as well as being able to reason about the hidden state of the game. The inspiration for using this architecture comes from Lample and Chaplot (2018) where they also used a CNN-LSTM.

3.1.2 Actor and Critic Networks

The actor and critic networks each use independent CNN-LSTM stacks to prevent shared-feature conflicts and to allow separate optimization dynamics. Sharing parameters would complicate credit assignment and update timing, as each network is trained for different objectives. The critic may

steer the CNN-LSTM towards features that assist with value estimation and the actor may steer the CNN-LSTM towards features that allow for improved policies. Both networks use their respective LSTM outputs as input for downstream modules.

The actor outputs an embedding vector, which is passed through an embedding layer to output a 5-action logits vector obtained via cosine similarity between the learned action embeddings and a 4-dimensional embedding per action. Additionally, the actor outputs mean and standard deviation parameters for a Gaussian distribution over 2-dimensional spatial coordinates. Originally, the means and standard deviations were determined with a "means" head and "standard deviations" head, though the x and y coordinates were extremely coordinated, resulting in separating the mean and standard deviation outputs into 4 heads.

Actions are sampled via softmax (for discrete decisions) and Gaussian sampling (for continuous positions), introducing stochasticity to encourage exploration. Because of this inherent stochasticity due to the entropy, the agent would explore the state space with this stochastic policy. Fully connected layers are used for all heads of the actor.

The critic produces a scalar Q-value by passing its CNN-LSTM output through three fully connected layers of size 134, 128, and 1, with ReLU activations in between.

3.2 Training Methods

3.2.1 Frame Skipping and Temporal Sequences

To reduce computational load, the agent chooses an action every 6 frames (200 ms at 30 FPS), following a frame skip strategy inspired by prior works Mnih et al. (2013); Lample and Chaplot (2018); OpenAI et al. (2019). Though each of the aforementioned papers used a frame skip of 4, I believe that a frame skip of 6 is sufficient enough to allow the agent to adequately perform. Additionally, the average reaction time of a human is around 200 milliseconds, meaning that with the game running at 30 frames-per-second, 6 frames per action results in an action every 5th of a second, mimicing the reaction speed of an average human. Each training sample spans 12 consecutive frames, helping the model learn temporal credit assignment. These overlapping sequences ensure continuity across samples, though the inclusion of a prior action can affect the ability to properly assign credit.

3.2.2 Soft Actor-Critic Optimization

I used the Soft Actor-Critic (SAC) algorithm Haarnoja et al. (2019), which optimizes an entropy-regularized objective using off-policy updates and a replay buffer. This is well-suited for self-play, where the opponent may change over time, since I am able to maintain a replay buffer of states thatplayer and the enemy may see. This allows for the policy to train on transitions that the enemy experiences as well.

To enable differentiable learning over discrete actions, I embed actions into a continuous space similar to token embeddings in NLP. This design allows gradient flow through the softmax selection mechanism, as otherwise, I would have to manipulate my gradient backpropagations, potentially using Gumbel-softmax to approximate gradients.

The actor loss is defined as

$$\mathcal{L}actor = \mathbb{E}_{s\sim\mathcal{D}}\left[\mathbb{E}_{a\sim\pi}\left[-Q_\theta(s,a) + \alpha\cdot\mathcal{H}(\pi(\cdot|s))\right]\right] \tag{1}$$

Where $Q_\theta(s,a)$ is the critic's estimate and $\mathcal{H}(\pi(\cdot|s))$ is the entropy of the policy. The states s are selected from a replay buffer $\mathcal{D}$ and the actions a are selected through querying the policy π. The temperature α is tuned automatically using the following equation:

$$\mathcal{L}_\alpha = \mathbb{E}_{s\sim\mathcal{D}}\left[-\alpha\cdot\mathcal{H}(\pi(s)) - \mathcal{H}_{target}\right] \tag{2}$$

α assists with matching the entropy of the policy with the target entropy. I chose to us e separate temperatures for the discrete and continuous action components since each component functions differently and has its own entropy. I used a target entropy of -5 for the discrete component since it is common to set the target entropy equal to the negative of the number of discrete actions. I had

trouble with determining a proper target entropy for my continuous component and ended up with a target entropy of 0, since my entropy continuously blew up when training.

I chose to use two critics trained using a standard Bellman backup to reduce the chance of overfitting and departing from true values.

$$\mathcal{L}critic = \frac{1}{N}\sum(Q_\theta(s,a) - y)^2 \tag{3}$$

$$y = r + \gamma(1-d)\min(Q_{target,1}(s',a'), Q_{target,2}(s',a')) \tag{4}$$

r represents the reward, d is a done flag, and γ is the discount factor. The critic loss is determined with a mean squared error loss between the critic's Q-value estimates and the expected return from using the Bellman backup and target critic estimates. Each critic is updated using their specific Q-value estimates and the minimum of the targed critic Q-values is taken to be conservative.

The target critics' parameters are updated using Polyak averaging. Each target critic is updated with the parameters of its corresponding critic.

$$\overline{\theta} \leftarrow \tau \cdot \theta + (1-\tau)\cdot\overline{\theta} \tag{5}$$

I used $\tau = 0.005$ and Adam for optimization, with a learning rate of 1×10^{-3} for the CNN-LSTM modules and the critics and a learning rate of 1×10^{-5} for the actor.

3.2.3 Replay Buffer

The replay buffer stores 50,000 transitions due to memory constraints. This capacity captures approximately 300 full game rollouts and provides sufficient diversity for effective off-policy training. Each episode runs for 1080 environment time steps. Since an action is taken every 6 frames, the replay buffer is able to store 300 full game rollouts with maximum episode length.

3.2.4 Self-Play Curriculum

I structured self-play through updating the enemy at an infrequent number of timesteps. I originally chose 100,000 time steps, but it proved to be too few, as the player itself was having difficulty learning the game. Even so, my methodology is such that only the player trains with both player and enemy transitions for a set number of steps with a frozen enemy policy. The enemy policy is then updated and then the player can continue training. This method of self-play is similar to the self-play utilized by OpenAI et al. (2019), though with a single past policy rather than a league of past policies. Nevertheless, this method of freezing the enemy policy ensures gradual progression and mitigates training instabilities.

3.3 Training Setup

Training occurs in real time, with the critic updated every action step using a batch size of 256. The actor is updated every other step that the critic is updated to reduce the chance for instabilities in training. The batches are shared between the actor and critic when updating, and training starts when I collect 10 times the batch size in the replay buffer to reduce correlation within batches. I used 40,000 timesteps of pretraining for the critic to reduce early instability in value estimation due to a rapidly changing policy.

4 Results and Discussion

Due to the complexity of both the model architecture and the game environment, training proved to be extremely challenging due to instabilities lurking at every corner. When first training, one particularly puzzling behavior emerged consistently during evaluation. Both agents would often travel to a corner of the map and engage in combat there. This was unexpected, as the environment only penalized leaving the center—not incentivizing the corners in any way. Figure 2 shows an example of this behavior. Initially, I suspected that the issue might stem from vanishing gradients or a flawed coordinate mapping between the network outputs and the game's pixel space. However, this theory

was contradicted by the agent's consistent ability to place both movement targets and damage zones at matching positions, with the chosen corner varying across different episodes. I later determined that in my implementation, I set the policy of the player and enemy to be the same, sharing references rather than values, meaning that policy updates to the player also affected the policy of the enemy when the enemy's policy was supposed to be frozen. Even so, an additional surprising outcome was that the agent stopped using its projectile ability altogether, which makes sense since projectiles spawned just outside the hitboxes of both the player and the enemy, meaning that the return of casting the ability was 0.

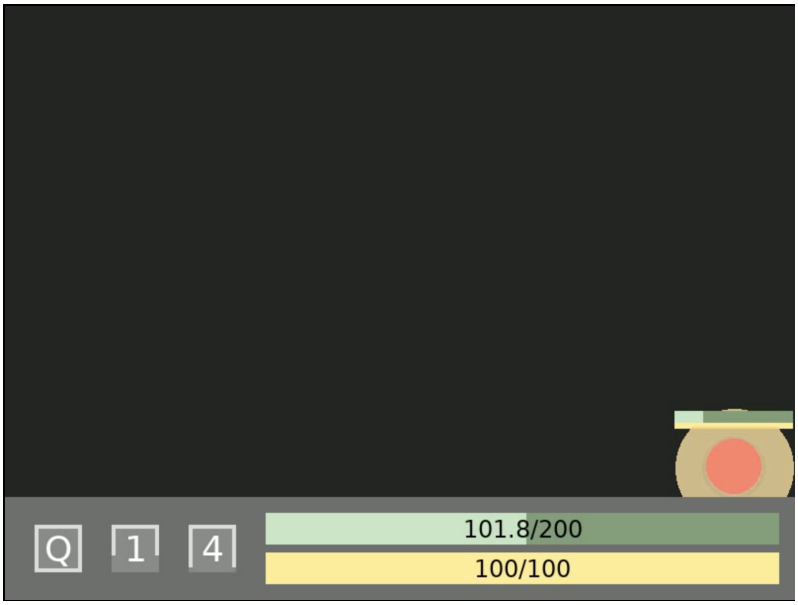


Figure 2: Both agents fighting in the bottom corner.

Due to the inherent complexity of the game, I progressively reduced the complexity of the action space to simplify training and gain more control over the agent's learning dynamics. Eventually, the player was only able to fire projectiles and the enemy agent was made entirely passive. Furthermore, I enabled full stamina to prevent stamina limitations from hindering learning. These changes significantly reduced the dimensionality of both the state and action spaces. Despite this enormous simplification, the setup still fulfilled the project's core goal, as it demonstrates that a visual input pipeline using CNN-LSTM architecture, coupled with Actor-Critic reinforcement learning and a self-play curriculum, can yield an effective policy.

4.1 Quantitative Evaluation

Under the restricted training setup, the trained agent significantly outperformed a randomly acting policy. In head-to-head evaluations, the trained policy consistently defeated the opponent in a matter of seconds, while the random agent was unable to secure a kill even after a full minute. **??** summarizes these results. The trained agent achieved an average return of 295 across 10 games, compared to only 44 for the random policy, meaning that the player was able to use the visual input to focus projectile shots towards the enemy.

Policy	Average Returns	Time to Kill (s)
Random	44	12
Trained	295	N/A

Table 1: Average returns and time to kill for random and trained policies.

Figure 3 and Figure 4 show the actor and critic losses, respectively. Interestingly, while the actor loss steadily decreases over time, the critic loss increases, meaning that the critic is struggling to provide stable Q-value estimates. This instability can introduce high variance into the policy gradient

updates, making training more erratic. Given that the actor and critic are mutually dependent, these instabilities can compound over time. Nevertheless, despite these large loss values, the agent was still able to perform remarkably well in practice.

It is important to note that decreasing actor losses are generally a good sign as oftentimes it shows that the agent is becoming more confident in the actions that it takes, as it tries to maximize entropy while simultaneously maximizing Q-values. Additionally, the increasing critic losses are not necessarily a terrible outcome, since with a large replay buffer that potentially consists of subpar trajectories, the critic may not be able to accurately predict the outcomes since it has become more overfitted to the optimal policy. Lastly, the larger Q-values predicted by the critic directly affect the actor loss as seen in (1).

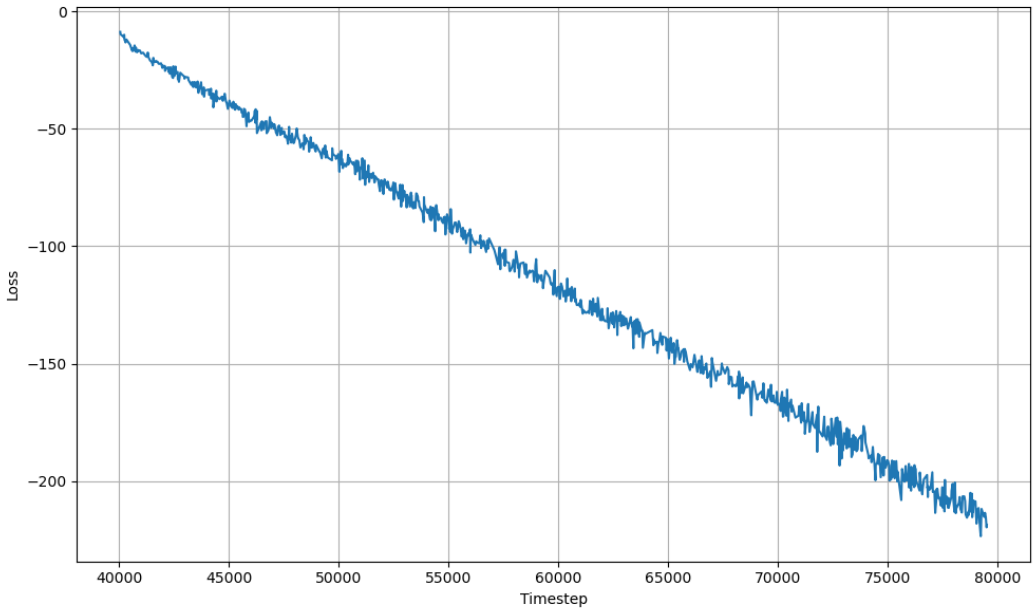


Figure 3: Actor loss across timesteps.

The returns over time, shown in Figure 6, quantitatively highlight the agent's effectiveness. In this setup, agents have 200 health and regenerate at a rate of roughly 1 HP per second. An additional 100-point reward is granted for winning a game. Given a game length of about 12 seconds, the theoretical reward ceiling is 310 (maximum damage plus victory bonus). The trained agent often reaches this limit, indicating that it deals near-maximal damage in minimal time, even while actor and critic losses remain high. This unexpected performance suggests that either surprisingly competent behavior can emerge from unstable training signals or that the policy fails to adjust once it reaches a saturation point of actor and critic losses.

Despite the agent's strong empirical performance, I remain uncertain how it achieves such results in the face of noisy and poorly converged Actor-Critic losses. It's possible that some emergent behavior or implicit regularization is helping the agent settle into high-reward strategies, even if those strategies aren't grounded in well-calibrated value estimates.

4.2 Qualitative Analysis

The ramifications of my research are minimal, as I have been unable to replicate these results with random seeds and it seems to be that the agent stumbled into the proper policy by sheer coincidence. Furthermore, the agent managed to break aspects of the game that I had built, allowing it to fire many more projectiles than initially thought possible. It was because of being able to shoot more projectiles that the agent was able to quickly dispatch the opponent. Figure **??** shows this incredible policy in action.

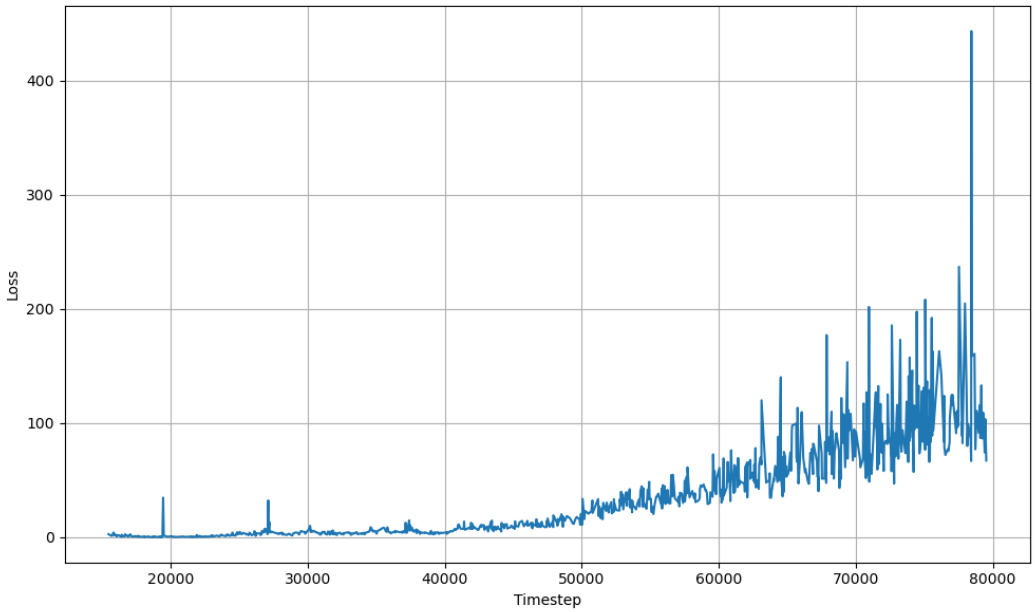


Figure 4: Critic loss across timesteps.

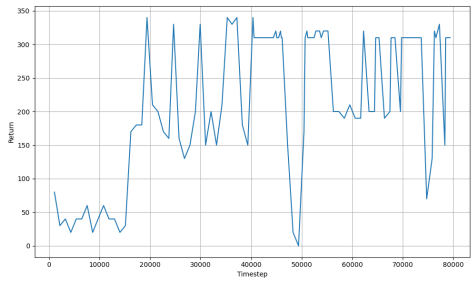


Figure 5: Returns over timesteps.

Once I managed to limit the ability for the agent to spam projectiles, I was unable to train a policy that performed as similarly. Even so, I am quite surprised that I was able to train a policy that used visual input to control the actions, specifically projectile positioning, of this agent. I was having extreme difficulty with training my agent and was incredibly overjoyed to find a policy that at least killed the agent within the maximum episode length.

5 Conclusion

This work demonstrates potential viability of integrating multiple neural network architectures, specifically convolutional neural networks, long short-term memory networks, and Actor-Critic methods, in training an agent to succeed in maximizing returns. Although my final policy operates within a reduced state and action space, the environment still provides sufficient complexity to gauge the effectivity of the policy, and serves as a rich environment for an agent to capture key aspects of spatial and temporal reasoning.

The results underscore challenges inherent in coordinating multiple neural networks. Training instabilities were an incredibly major obstacle, likely due to the fragile stabilities of the interconnected networks with shifting objectives. Nevertheless, even with these challenges, the agent was able to learn a policy that consistently maximized returns within the specified time horizon.

Several directions remain for improving both stability and agent performance, as well as baselines for comparison. I would like to increase the complexity of the game through progressively adding

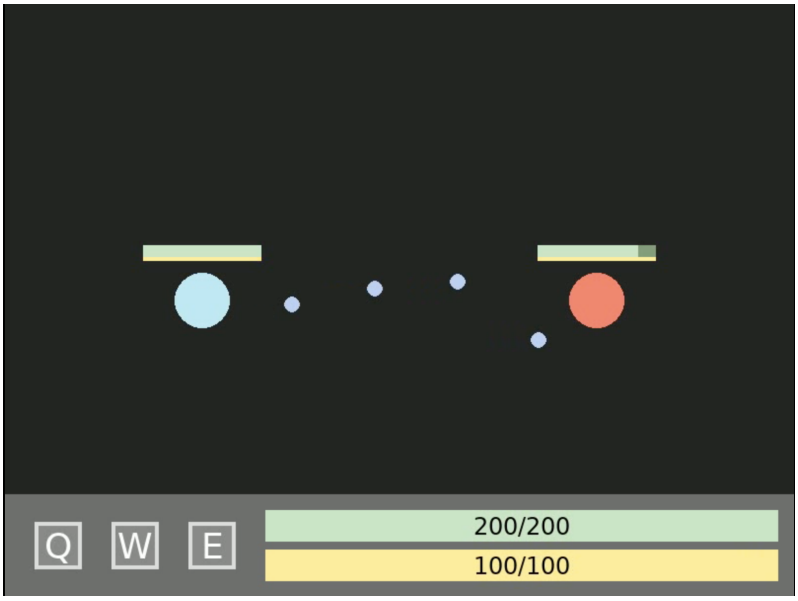


Figure 6: Many projectiles fired in a short period of time.

more actions and allowing the enemy to play. Additionally, I would like to update the enemy policy infrequently to encourage the player to discover more intricate strategies. The observed discrepancy between the actor and critic losses and the agent's in-game performance raises the suspicion that the simple environment allows high returns to be achieved despite "suboptimal" performance.

Future work would aim to establish an alternative baseline to random policies. Training the CNN-LSTM block and actor and critics separately and surgically combining the networks might prove to be a competent baseline for comparing the ability for an agent to learn with a pre-processed state and without one. Nonetheless, I have really enjoyed undertaking this project and am excited to continue advancing my understanding in this field.

6 Team Contributions

- **Joshua Boisvert** created game and programmed Soft Actor-Critic method, as well as training regime and wrote the report.

Changes from Proposal Instead of using the entire action space and training with self-play, I only manage to train within a limited action space and with no self-play. Even so, I was able to train a policy that consistently dispatched the enemy agent.

References

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. 2018. Emergent Complexity via Multi-Agent Competition. arXiv:1710.03748 [cs.AI] `https://arxiv.org/abs/1710.03748`

Debidatta Dwibedi and Anirudh Vemula. 2020. Playing Games with Deep Reinforcement Learning.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. 2019. Soft Actor-Critic Algorithms and Applications. arXiv:1812.05905 [cs.LG] `https://arxiv.org/abs/1812.05905`

Taewon Kim, Yeseong Park, Youngbin Park, and Il Hong Suh. 2020. Acceleration of Actor-Critic Deep Reinforcement Learning for Visual Grasping in Clutter by State Representation Learning Based on Disentanglement of a Raw Input Image. arXiv:2002.11903 [cs.LG] `https://arxiv.org/abs/2002.11903`

Guillaume Lample and Devendra Singh Chaplot. 2018. Playing FPS Games with Deep Reinforcement Learning. arXiv:1609.05521 [cs.AI] https://arxiv.org/abs/1609.05521

Sascha Lange, Martin Riedmiller, and Arne Voigtländer. 2012. Autonomous reinforcement learning on raw visual input data in a real world application. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN.2012.6252823

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

OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. arXiv:1912.06680 [cs.LG] https://arxiv.org/abs/1912.06680

A Additional Experiments

I did not have time to finish, but I started working on a separate training method in which I trained the CNN-LSTM modules and the Critic and Actor modules simultaneously with losses for state prediction for the CNN-LSTM modules and true states for the Critic and Actor modules. I was able to reduce the losses for my Critics and CNN-LSTM significantly, but was unable to figure out entropy struggles related to the Actor. My policy consistently chose positions that were in the corners due to trying to maximize entropy.