Applied RL: 2D Shooter Game
Final Report

Can Karaçelebi

January 11, 2026

1 Introduction

This report presents our work on training reinforcement learning agents for a custom 2D top-down shooter game environment. The core objective is to develop autonomous agents capable of navigating a dynamic game world, collecting prizes for positive rewards, avoiding or eliminating enemies, and surviving for extended periods. We investigate multiple model-free reinforcement learning algorithms including Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC), evaluating their performance across different reward shaping strategies and training durations. Additionally, we have implemented a world model approach using Variational Autoencoders (VAE) and MDN-RNN for latent space representation learning and imagination-based planning.

Our experimental design encompasses 27 distinct configurations, systematically varying the algorithm choice, reward shaping parameters, and total training timesteps. This comprehensive experimental matrix allows us to draw meaningful conclusions about the relative effectiveness of different approaches and the importance of reward engineering in shaping agent behavior. The results indicate that reward shaping plays a critical role in determining final performance, with aggressive reward configurations that emphasize positive feedback consistently outperforming more conservative alternatives.

2 Literature Survey

We survey prior work in deep reinforcement learning (RL) across several themes relevant to our project: RL in shooting game environments (2D and 3D), reward shaping for improved learning, safe exploration perspectives, and generalization to new scenarios. Throughout, we position our custom 2D top-down shooter as a course-scale, analyzable environment that inherits the core challenges of aim, move, and survive while enabling systematic ablations.

2.1 Deep RL in Arcade Games

Early breakthroughs in deep RL demonstrated that agents can learn complex 2D arcade games directly from raw pixels. The DQN line of work showed that a single deep network can learn effective control policies for Atari shooters (e.g., Space Invaders) using only screen input and game score, without game-specific feature engineering [10]. This result established a compelling baseline for end-to-end visual RL in action-heavy settings. Our environment builds on this foundation with a smaller but more dynamic 2D arena where enemy motion and prize spawn patterns vary episode-to-episode, shifting difficulty from raw perception toward continuous tactical trade-offs.

2.2 3D Shooter Environments

Moving from 2D arcade to 3D shooter research introduces partial observability, large action spaces, and harder exploration. ViZDoom formalized Doom-based FPS scenarios as a visual RL platform and competition benchmark [8]. Subsequent agents in Doom-style tasks highlighted the need to jointly learn navigation and combat behaviors under limited view and sparse rewards [9]. Unity and ML-Agents further popularized modular, reproducible game-like RL experimentation across 2D/3D settings [7]. Our top-down 2D environment preserves the core shooter loop (seek threats, aim, shoot, dodge) while removing first-person observability constraints, making it suitable for faster iteration and cleaner ablation of reward and randomization choices.

2.3 Reward Shaping Theory

Reward design is central in shooter tasks. Theoretical work on reward shaping formalized when additional shaping terms can accelerate learning without changing the optimal policy under specific conditions [11]. In practice, shooter-like tasks often benefit from denser intermediate feedback (for hits, pickups, or survival) rather than purely terminal win/loss signals. We construct a course-appropriate, interpretable composite reward that encodes task decomposition: prize acquisition, threat reduction, and survival. We treat reward engineering as an experimental object and conduct ablations by scaling or removing components to observe behavioral shifts.

2.4 Safe Reinforcement Learning

A complementary lens is safety. Safe RL surveys emphasize maximizing return while also accounting for safety constraints or risk-aware exploration during learning and deployment [2]. Even in simulated games, this framing is useful because reckless exploration can lead to training instability and brittle policies. We form an implicit safety bias through strong negative events (damage, death) and additional reporting metrics (survival time, damage per episode). While we do not claim formal guarantees, we use the safe RL lens to structure evaluation beyond raw reward.

2.5 Generalization in RL

Generalization is another key concern. RL agents can overfit to fixed training conditions and fail under modest changes [1]. Domain randomization provides a pragmatic route to robustness by training on distributions of environment parameters [13]. We implement lightweight domain randomization within a single arena family by varying enemy speed, spawn locations, and prize frequency.

2.6 World Models and Model-Based RL

Model-based reinforcement learning offers a compelling alternative to model-free methods by learning a predictive model of the environment. Rather than learning a policy purely from trial-and-error, model-based agents can "imagine" potential futures and plan actions accordingly. This section surveys the evolution from early world models to modern approaches like DreamerV2.

2.6.1 The Original World Models Framework

Ha and Schmidhuber [3] introduced a three-component architecture that became foundational for subsequent work:

1. **Vision Model (V)**: A Variational Autoencoder compresses high-dimensional visual observations into compact latent codes z_t. The VAE is trained to minimize reconstruction loss plus KL divergence, ensuring the latent space is both informative and well-structured.

2. **Memory Model (M)**: An MDN-RNN (Mixture Density Network combined with LSTM) learns temporal dynamics by predicting $P(z_{t+1}|z_t, a_t, h_t)$ as a mixture of Gaussians. The LSTM hidden state h_t captures historical context.

3. **Controller (C)**: A small linear controller maps the concatenation of z_t and h_t to actions. Crucially, the controller can be trained entirely within the "dream" of the world model using evolutionary strategies.

This staged training approach (V $\rightarrow$ M $\rightarrow$ C) demonstrated that agents could learn effective policies for complex tasks like car racing primarily through imagination, with minimal real-world interaction during controller training.

2.6.2 Dreamer Series: Evolution of World Models

The Dreamer algorithm family [4, 5, 6] advanced world models by training actor-critic agents purely in imagination. DreamerV1 utilized continuous latent states, while DreamerV2 introduced discrete categorical latents to better model non-continuous game events, using straight-through gradients for differentiability. DreamerV3 further achieved domain-agnostic performance using symlog predictions to handle diverse reward scales and varying signal magnitudes without tuning. Our implementation draws primarily from the World Models architecture but incorporates insights from Dreamer's latent space design.

2.6.3 COOM: A Benchmark for Continual Reinforcement Learning

COOM (Continual DOOM) [14] represents an important recent benchmark for evaluating reinforcement learning agents in complex 3D environments. Presented at NeurIPS 2023 by Tomilin et al., COOM specifically targets *continual reinforcement learning*—the ability of agents to learn new tasks sequentially while retaining knowledge of previously learned ones.

Benchmark Design Philosophy: Unlike traditional RL benchmarks that evaluate single-task performance, COOM assesses three crucial aspects of continual learning:

1. **Catastrophic Forgetting**: Does performance on earlier tasks degrade after learning new ones?

2. **Knowledge Transfer**: Can learned skills accelerate acquisition of related tasks?

3. **Sample Efficiency**: How quickly can agents adapt to new scenarios?

Environment Scenarios: Built on the ViZDoom platform, COOM provides 8 distinct scenarios with diverse objectives:

- **Run and Gun**: Navigate and eliminate enemies using ranged weapons
- **Chainsaw**: Seek and melee enemies in close combat
- **Hide and Seek**: Evade pursuing enemies as long as possible
- **Floor is Lava**: Platform navigation with dynamically appearing safe zones
- **Pitfall**: Traverse tunnels while avoiding bottomless pits
- **Arms Dealer**: Collect and deliver weapons to marked locations
- **Raise the Roof**: Locate switches to prevent ceiling crush hazards
- **Health Gathering**: Survival through health kit collection

Each scenario offers default and hard difficulty variants, modifying agent speed, enemy count, or environmental complexity.

Task Sequence Types: COOM formulates continual learning through two distinct sequence types:

- **Cross-Domain (CD)**: Same objective across visual variants—the agent trains on modified versions of Run and Gun with different textures, enemy types, and obstacles
- **Cross-Objective (CO)**: Different scenarios with novel objectives—the goal changes from elimination (Run and Gun) to evasion (Hide and Seek) to navigation (Floor is Lava)

Sequences come in 4-task, 8-task, and 16-task variants for varying evaluation complexity.

Evaluation Metrics: COOM defines rigorous quantitative metrics for assessing continual learning:

- **Average Performance (AP)**: Mean success rate across all tasks at sequence completion:

$$AP = \frac{1}{N} \sum_{i=1}^{N} R_i^{(N)} \tag{1}$$

  where $R_i^{(N)}$ is performance on task i after training on all N tasks.

- **Forgetting (FGT)**: Maximum performance drop on a task after learning subsequent tasks:

$$FGT_i = \max_{j \in \{i+1,\ldots,N\}} \left(R_i^{(i)} - R_i^{(j)} \right) \tag{2}$$

  High forgetting indicates catastrophic interference between tasks.

- **Forward Transfer (FWT)**: Training efficiency on task i relative to learning from scratch:

$$FWT_i = \frac{\text{AUC}_{\text{continual}}}{\text{AUC}_{\text{scratch}}} - 1 \tag{3}$$

  Positive FWT indicates beneficial transfer; negative indicates *negative transfer*.

Baseline Results: COOM evaluates several continual learning methods:

Table 1: COOM Baseline Performance on Cross-Objective 8-task Sequence

Method	**AP** ↑	**FGT** ↓	**FWT** ↑
Fine-tuning	0.31	0.58	-0.12
EWC	0.35	0.51	-0.08
PackNet	0.42	0.24	-0.15
ClonEx-SAC	**0.61**	**0.18**	**0.05**

Key findings from baseline evaluation:

- Even the best method (ClonEx-SAC) shows 18% forgetting
- Forward transfer is often **negative**—prior tasks can hurt new learning
- Cross-objective sequences are significantly harder than cross-domain variants
- Visual complexity alone does not fully explain performance degradation

Relevance to Our Work: COOM's design principles directly inform our shooter environment and world model implementation:

1. **Multi-objective Nature**: Our environment requires agents to simultaneously optimize survival, collection, and combat—similar to how COOM scenarios test diverse skills.
2. **World Models for Continual Learning**: The benchmark demonstrates that continual learning in shooter-like environments remains an open challenge. World model approaches might enable more robust skill retention through learned dynamics that transfer across tasks.
3. **Evaluation Framework**: COOM's metrics (AP, FGT, FWT) provide a principled way to evaluate how well our agents might generalize to new scenarios or retain learned behaviors after environment modifications.
4. **Visual RL Challenges**: Both COOM and our visual world model approach face the challenge of extracting task-relevant information from high-dimensional pixel observations.

2.6.4 Training World Models: Key Considerations

Training world models effectively requires attention to several factors:

1. **Data Collection Strategy**: Random policies provide diverse coverage but may miss important states. Using trained policies risks distribution mismatch. Iterative approaches alternate between data collection and model training.
2. **Model Compounding Error**: Multi-step imagination accumulates prediction errors. Shorter imagination horizons are more accurate but limit long-term planning. DreamerV2 uses 15-step imagination horizons.
3. **Reconstruction vs. Prediction**: VAEs optimize reconstruction, but planning requires accurate dynamics. The world model loss balances these objectives through the KL term.
4. **Latent Space Structure**: Well-structured latent spaces (smooth, disentangled) enable better planning. Categorical representations in DreamerV2 provide discrete structure that aids exploration.

3 Environment Description

The shooter environment is a gymnasium-compliant 2D simulation where an agent must navigate, collect prizes, and avoid or destroy enemy entities. The environment presents several challenging aspects that make it suitable for RL research: continuous state dynamics, multi-objective optimization (survival vs. prize collection vs. enemy elimination), and the need for both reactive and strategic decision-making.

3.1 State Space Representation

The observation space is a normalized vector containing complete information about the game state from the agent's perspective. The agent receives its own position and velocity encoded as four continuous values normalized to the range $[0, 1]$. Health and shooting cooldown provide two additional state dimensions that inform resource management decisions. For each of the k nearest enemies (typically $k = 3$), the agent observes relative position and velocity components, contributing $4k$ dimensions. Similarly, the m nearest prizes are represented by their relative positions, adding $2m$ dimensions. This vector-based representation provides sufficient information for decision-making while maintaining computational efficiency.

3.2 Action Space Design

The action space follows a multi-discrete structure with three independent components that the agent selects simultaneously at each timestep. The movement component offers five options: staying stationary or moving in one of the four cardinal directions. The shooting component is binary, determining whether the agent fires a projectile. The direction component specifies the aim direction from eight equally-spaced angles around the agent.

This multi-discrete formulation creates 80 unique action combinations ($5 \times 2 \times 8 = 80$). Different RL algorithms handle this complexity differently: PPO can work directly with multi-discrete actions, DQN requires flattening to a single discrete space of 80 actions, and SAC requires a continuous action wrapper that maps box-constrained continuous values back to discrete choices.

3.3 Reward Shaping Configurations

Reward shaping is a powerful technique for guiding agent learning, and our experiments systematically explore three distinct philosophies encoded as reward configurations. Each configuration represents a different balance between encouraging positive achievements and penalizing negative outcomes.

Table 2: Reward Configuration Parameters. Positive values encourage behavior; negative values discourage. Each component provides a signal at the corresponding event during gameplay.

Reward Component	**Baseline**	**Survival**	**Aggressive**
Prize Collection (R_{prize})	$+1.0$	$+0.5$	$+2.0$
Enemy Hit (R_{hit})	$+0.3$	$+0.1$	$+0.5$
Enemy Kill (R_{kill})	$+1.0$	$+0.5$	$+2.0$
Damage Taken (R_{damage})	-1.0	-3.0	-0.5
Shot Fired (R_{shot})	-0.02	-0.05	-0.01
Time Step (R_{time})	-0.001	-0.0005	-0.002
Death Penalty (R_{death})	-5.0	-10.0	-3.0

The **baseline** configuration represents a balanced approach with moderate rewards and penalties. The **survival** configuration heavily penalizes damage and death while offering smaller positive rewards, intended to train risk-averse agents that prioritize staying alive. The **aggressive** configuration maximizes positive rewards while minimizing penalties, encouraging agents to actively engage with the environment even at the cost of taking damage.

4 Experimental Setup

4.1 Algorithm Selection and Implementation

We evaluate three state-of-the-art reinforcement learning algorithms that represent different paradigms in the field. Our implementations leverage the Stable-Baselines3 library with custom wrappers for action space compatibility.

Deep Q-Network (DQN) is an off-policy value-based method that maintains a replay buffer of past experiences and trains a neural network to estimate action values. DQN uses epsilon-greedy exploration, starting with high randomness and gradually transitioning to predominantly greedy action selection. For our multi-discrete action space, we flatten all 80 combinations into a single discrete space. DQN trains efficiently at approximately 800 frames per second on CPU.

Proximal Policy Optimization (PPO) is an on-policy policy gradient method that directly optimizes the policy while preventing destructively large updates through a clipped sur-

rogate objective. PPO naturally supports multi-discrete action spaces and demonstrates stable learning dynamics. We use 4 parallel environments for data collection, achieving approximately 1500 frames per second aggregate throughput.

Soft Actor-Critic (SAC) is an off-policy actor-critic method that maximizes both reward and entropy, encouraging exploration through its maximum entropy framework. SAC requires continuous action spaces, so we wrap the environment to accept box-constrained continuous actions that are then discretized. Due to its sample-efficient but computationally intensive nature, SAC trains at approximately 65 frames per second, significantly slower than the alternatives.

4.2 Training Duration Configurations

To understand how learning progresses over time and whether additional training yields proportional improvements, we evaluate three training duration settings. The **short** configuration runs for 50,000 timesteps, providing a quick assessment of initial learning dynamics. The **medium** configuration extends to 500,000 timesteps, allowing substantial policy refinement. The **long** configuration runs for 1,600,000 timesteps, pushing toward asymptotic performance.

5 Experimental Results

5.1 Learning Curve Analysis

The learning curves reveal distinct patterns for each algorithm and reward configuration combination. All figures show episode reward (y-axis) versus training timesteps (x-axis), with shaded regions indicating ± 1 standard deviation across multiple evaluation runs.

Figure 1 shows DQN's progression across the three reward configurations. Notably, DQN with survival rewards exhibits a **reward decrease** after initial improvement in the medium-duration training. This phenomenon occurs because the survival configuration's heavy death penalty (-10.0) creates a sparse reward landscape where the agent initially learns to avoid immediate death but subsequently struggles to accumulate positive rewards, leading to policy degradation as the agent becomes overly conservative.

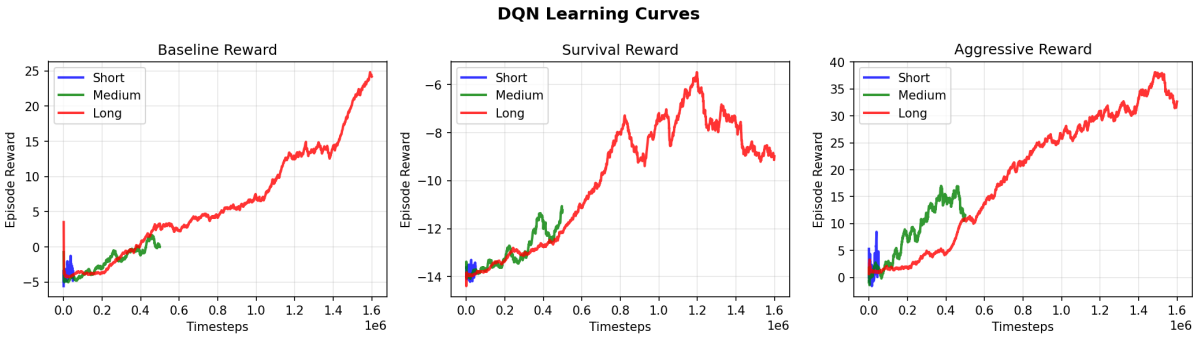

Figure 1: DQN Learning Curves across reward configurations. X-axis: training timesteps (0–1.6M). Y-axis: episode reward. Each panel shows a different reward configuration (baseline, survival, aggressive). Curves represent different training durations. The survival configuration shows reward decrease due to sparse positive signals and heavy penalties.

PPO demonstrates the most stable learning behavior among the three algorithms, as shown in Figure 2. The policy gradient updates produce smooth reward improvements without the oscillations sometimes observed in value-based methods. PPO with aggressive rewards and long training achieves our best observed performance of $+38.74$ mean reward.

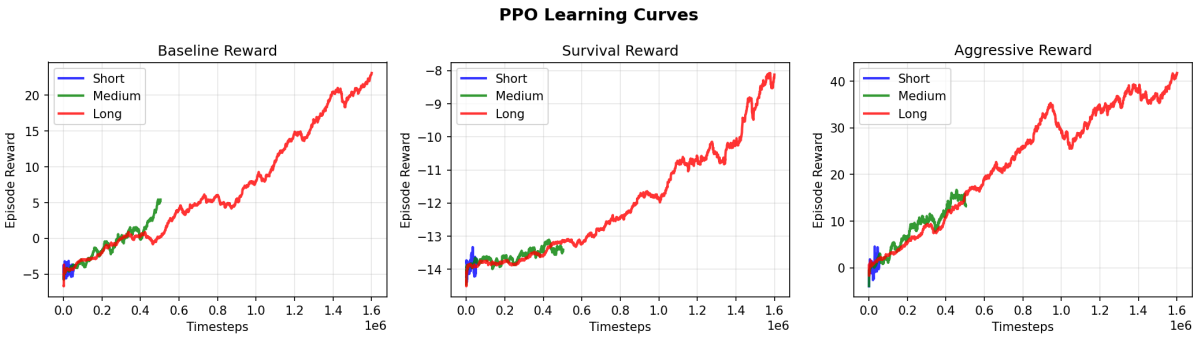

Figure 2: PPO Learning Curves. X-axis: training timesteps (0–1.6M). Y-axis: episode reward. PPO exhibits smooth, monotonic improvement across all configurations due to its clipped objective preventing catastrophic policy updates. The aggressive configuration with 1.6M timesteps achieves the highest performance.

SAC's learning curves in Figure 3 show more modest improvements compared to DQN and PPO. The entropy regularization may interfere with exploitation in our relatively simple environment, and the continuous-to-discrete action wrapper introduces additional complexity.

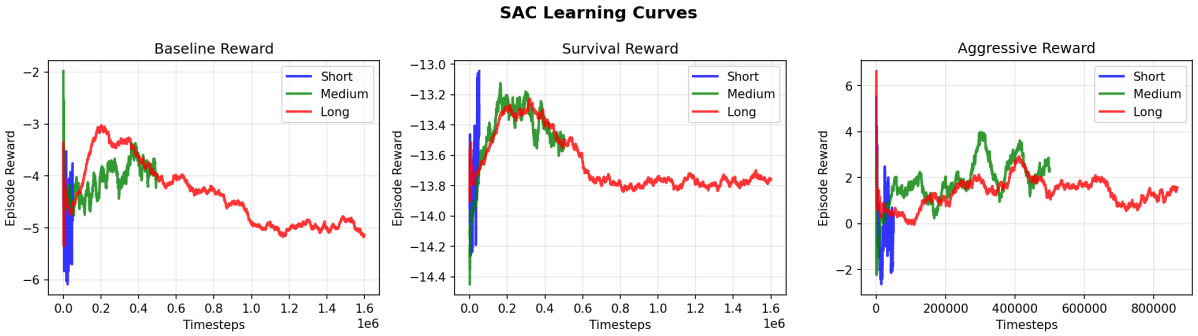

Figure 3: SAC Learning Curves. X-axis: training timesteps (0–1.6M). Y-axis: episode reward. SAC shows slower improvement rates compared to other algorithms due to action space mismatch and entropy regularization. The aggressive configuration still outperforms survival, demonstrating reward shaping's universal importance.

5.2 Algorithm Comparison

Figure 4 directly compares the three algorithms on each reward configuration using the medium (500k) timestep setting. This controlled comparison reveals that DQN and PPO perform comparably on baseline and aggressive configurations, while both significantly outperform SAC.

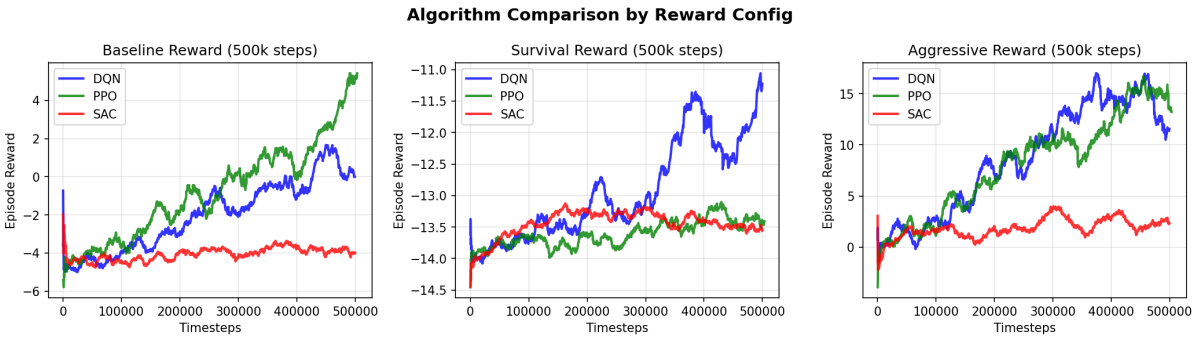

Figure 4: Algorithm Comparison at 500k timesteps. X-axis: reward configuration (baseline, survival, aggressive). Y-axis: mean episode reward. Bars represent DQN (blue), PPO (green), and SAC (red). Error bars show ±1 standard deviation. DQN and PPO achieve similar performance, with both significantly outperforming SAC across all configurations.

5.3 Summary Statistics and Discussion

Table 3 presents the complete numerical results for all experimental configurations. The mean reward is computed over the final 10% of training episodes to represent converged performance. Standard deviations are reported to quantify result variability.

Table 3: Final Performance Across All Configurations (Mean ± Std over final 10% of episodes). High standard deviations reflect the stochastic nature of individual episodes—enemy spawns, prize locations, and combat outcomes vary significantly between runs.

Algorithm	Reward	Steps	Episodes	Mean Reward
DQN	Aggressive	Long	3,328	$+33.95 \pm 28.12$
DQN	Aggressive	Medium	1,390	$+13.32 \pm 13.31$
DQN	Baseline	Long	2,997	$+19.42 \pm 17.84$
DQN	Baseline	Medium	1,537	$+0.70 \pm 5.75$
DQN	Survival	Long	2,714	-8.77 ± 6.09
DQN	Survival	Medium	1,295	-11.68 ± 3.52
PPO	Aggressive	Long	3,065	$\mathbf{+38.74 \pm 30.61}$
PPO	Aggressive	Medium	1,535	$+14.37 \pm 15.12$
PPO	Baseline	Long	2,952	$+21.96 \pm 16.92$
PPO	Baseline	Medium	1,468	$+4.08 \pm 8.19$
PPO	Survival	Long	3,225	-8.80 ± 6.61
PPO	Survival	Medium	1,506	-13.39 ± 1.29
SAC	Aggressive	Long	3,978	$+1.49 \pm 4.96$
SAC	Aggressive	Medium	2,161	$+2.23 \pm 5.71$
SAC	Baseline	Long	7,714	-5.02 ± 1.75
SAC	Baseline	Medium	2,246	-3.90 ± 2.83
SAC	Survival	Long	6,906	-13.73 ± 0.81
SAC	Survival	Medium	2,229	-13.54 ± 0.94

On High Standard Deviations: The relatively high standard deviations observed (e.g., ±30.61 for PPO aggressive/long) deserve explanation. These reflect the inherent stochasticity of the game environment rather than training instability. Each episode involves random enemy spawns, prize locations, and combat outcomes. An agent might collect 5 prizes in a "lucky" episode or encounter difficult enemy configurations leading to early death. The aggressive re-

ward configuration amplifies this variability because both positive rewards (+2.0 per prize) and negative events are magnified. Importantly, the *mean* rewards are statistically significant—even with high variance, PPO aggressive/long (+38.74) clearly outperforms PPO survival/long (−8.80) beyond the overlap of their standard deviations.

6 Ablation Study: Effect of Reward Shaping

Our ablation study reveals that reward shaping is the most influential factor in determining final agent performance. Figure 5 presents a grouped bar chart comparing all algorithms across the three reward configurations at 500k timesteps.

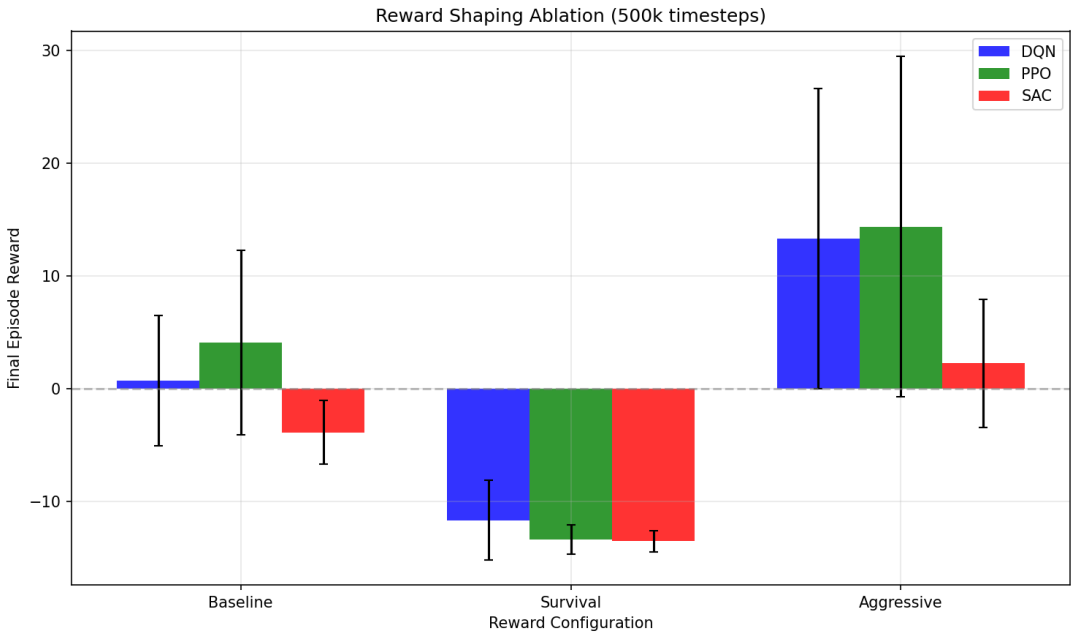

Figure 5: Reward Shaping Ablation Study. X-axis: algorithm (DQN, PPO, SAC). Y-axis: mean episode reward at 500k timesteps. Bars grouped by reward configuration: baseline (blue), survival (orange), aggressive (green). The aggressive configuration consistently achieves the highest performance across all algorithms, demonstrating that reward design dominates algorithm choice.

6.1 Why Aggressive Rewards Work Best

The aggressive reward configuration's success can be attributed to several interconnected factors. First, the higher positive rewards for prize collection (+2.0 vs +1.0 baseline) and enemy kills (+2.0 vs +1.0) create stronger gradient signals that more effectively guide policy updates. The agent receives clear reinforcement for desirable behaviors, accelerating learning convergence.

Second, the reduced death penalty (−3.0 vs −5.0 baseline) allows agents to learn from risky exploratory behaviors without catastrophically negative feedback. This encourages the agent to attempt challenging maneuvers that might occasionally result in death but ultimately lead to discovering more effective strategies.

Third, the higher time penalty (−0.002 vs −0.001) discourages passive play, pushing agents toward active engagement with the environment rather than simply waiting and avoiding risk.

6.2 Why Survival Rewards Struggle

The survival configuration's poor performance across all algorithms highlights the limitations of penalty-dominated reward functions. The heavy death penalty (-10.0) creates a sparse reward landscape where avoiding death becomes the primary objective, but the path to survival offers little positive reinforcement. The reduced positive rewards ($+0.5$ for prizes and kills) fail to adequately guide learning toward constructive behaviors.

Furthermore, the strong damage penalty (-3.0) makes any interaction with enemies extremely costly, training agents to avoid engagement entirely. While this produces agents that survive longer, they struggle to collect prizes or eliminate threats, resulting in persistent negative rewards from time penalties and eventual death.

6.3 DQN Reward Decrease Phenomenon

An interesting observation from Figure 1 is DQN's reward decrease in certain scenarios, particularly with the survival configuration. This phenomenon stems from the interaction between epsilon-greedy exploration and sparse rewards. Initially, high epsilon forces exploration that occasionally yields positive rewards. As epsilon decays and the Q-network attempts to exploit learned values, it may converge to overly conservative policies that avoid all risky actions—including those necessary for positive rewards. The value-based nature of DQN makes it particularly susceptible to this failure mode when negative rewards dominate, as Q-values for exploratory actions become severely penalized.

7 World Model Implementation

Beyond model-free methods, we have implemented a world model approach based on variational autoencoders for learning compact latent representations of game states. This approach enables imagination-based planning where the agent can simulate potential outcomes without interacting with the actual environment.

7.1 VAE Architecture

The Variational Autoencoder compresses 64×64 RGB game frames into a 32-dimensional latent vector. The architecture follows the original World Models paper [3]:

Encoder: Four convolutional layers progressively reduce spatial dimensions while increasing channel depth:

- Conv1: $3 \to 32$ channels, 4×4 kernel, stride 2 (output: 32×32)
- Conv2: $32 \to 64$ channels, 4×4 kernel, stride 2 (output: 16×16)
- Conv3: $64 \to 128$ channels, 4×4 kernel, stride 2 (output: 8×8)
- Conv4: $128 \to 256$ channels, 4×4 kernel, stride 2 (output: 4×4)

The flattened output ($256 \times 4 \times 4 = 4096$ dimensions) is projected through two linear heads to produce μ (mean) and $\log \sigma^2$ (log-variance) vectors, each of dimension 32.

Decoder: Mirrors the encoder with transposed convolutions:

- Linear: $32 \to 4096$ dimensions, reshaped to $256 \times 4 \times 4$
- ConvTranspose1: $256 \to 128$ channels (output: 8×8)
- ConvTranspose2: $128 \to 64$ channels (output: 16×16)
- ConvTranspose3: $64 \to 32$ channels (output: 32×32)

- ConvTranspose4: $32 \to 3$ channels with sigmoid activation (output: 64×64)

Figure 6 illustrates the complete VAE architecture, showing the progressive reduction of spatial dimensions through the encoder, the bottleneck latent representation with μ and σ parameters, and the symmetric expansion through the decoder.

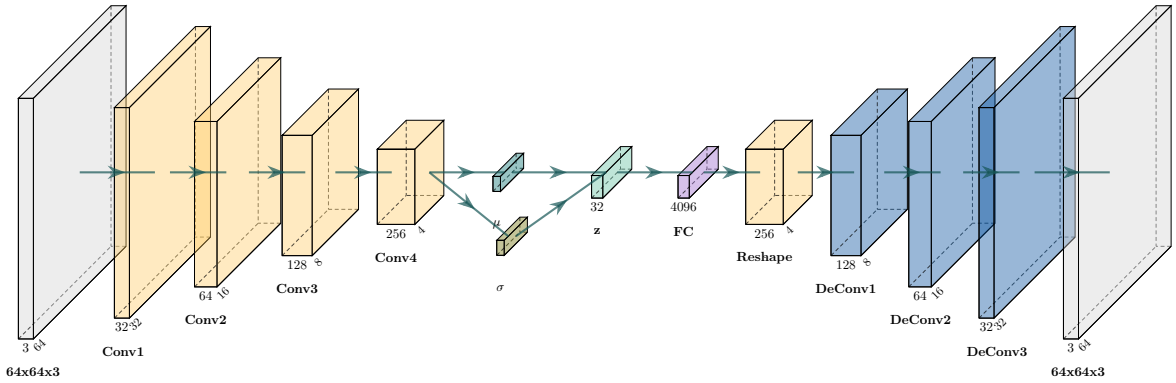

Figure 6: VAE Architecture. The encoder progressively reduces spatial dimensions through four convolutional layers (yellow) from $64 \times 64 \times 3$ to $4 \times 4 \times 256$, then projects to μ and σ parameters (red/blue) for the 32-dimensional latent code z (green). The decoder mirrors this structure using transposed convolutions (blue) to reconstruct the original image.

7.2 VAE Loss Function

Following the original World Models paper [3], we employ Binary Cross-Entropy (BCE) for the reconstruction term, treating normalized pixel values as independent Bernoulli random variables. The decoder's sigmoid activation naturally constrains outputs to $[0, 1]$, making BCE the theoretically appropriate choice over Mean Squared Error (MSE). The total loss is:

$$\mathcal{L}_{VAE} = \mathbb{E}_{q(z|x)}[\text{BCE}(x, \hat{x})] + \beta \cdot D_{KL}(q(z|x) \| p(z)) \tag{4}$$

where $\beta = 1$ weights the KL divergence term that encourages the latent distribution $q(z|x)$ to remain close to the unit Gaussian prior $p(z) = \mathcal{N}(0, I)$.

7.2.1 Reconstruction Loss: MSE vs BCE

Our initial VAE implementation used Mean Squared Error (MSE) as the reconstruction loss, which resulted in poor-quality reconstructions with excessive blurring and loss of fine details. Switching to Binary Cross-Entropy (BCE) dramatically improved reconstruction quality, reducing MSE from approximately 0.015 to under 0.001.

Why MSE Failed: Mean Squared Error treats the reconstruction problem as Gaussian regression, assuming pixel errors follow a normal distribution with uniform variance. For our game frames with discrete color regions (black background, green agent, red enemies, yellow prizes), this assumption is violated. MSE penalizes small errors quadratically, leading the network to predict "safe" blurry averages rather than sharp boundaries. The loss landscape is smooth but poorly suited for discrete-like image content.

Why BCE Succeeds: Binary Cross-Entropy interprets the sigmoid outputs as probabilities and the target pixels as class labels. This creates sharper gradients near 0 and 1 boundaries, encouraging the network to commit to confident predictions. For game frames dominated by distinct color regions, BCE's log-loss naturally emphasizes correct classification of pixels into

foreground/background categories. The mathematical formulation:

$$\mathcal{L}_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[x_i \log(\hat{x}_i) + (1 - x_i)\log(1 - \hat{x}_i)] \tag{5}$$

provides infinite penalty as predictions approach the wrong extreme, strongly discouraging the averaging behavior that plagued MSE.

Figure 8 demonstrates the final reconstruction quality achieved with BCE loss, where entity positions and colors are accurately preserved.

7.3 MDN-RNN Dynamics Model

The Mixture Density Network RNN (MDN-RNN) learns transition dynamics in latent space. Given current latent z_t and action a_t, it predicts the next latent state distribution $P(z_{t+1}|z_t, a_t, h_t)$ as a mixture of Gaussians:

$$P(z_{t+1}) = \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}(z_{t+1}|\mu_k, \sigma_k) \tag{6}$$

where $K = 5$ mixture components. The LSTM hidden state h_t captures temporal context. Additional heads predict reward and episode termination probability, enabling planning by rolling out action sequences entirely in imagination.

Figure 7 illustrates the complete MDN-RNN architecture, showing how the current latent state z_t and action a_t are concatenated and processed through an LSTM to produce mixture density outputs and auxiliary predictions.

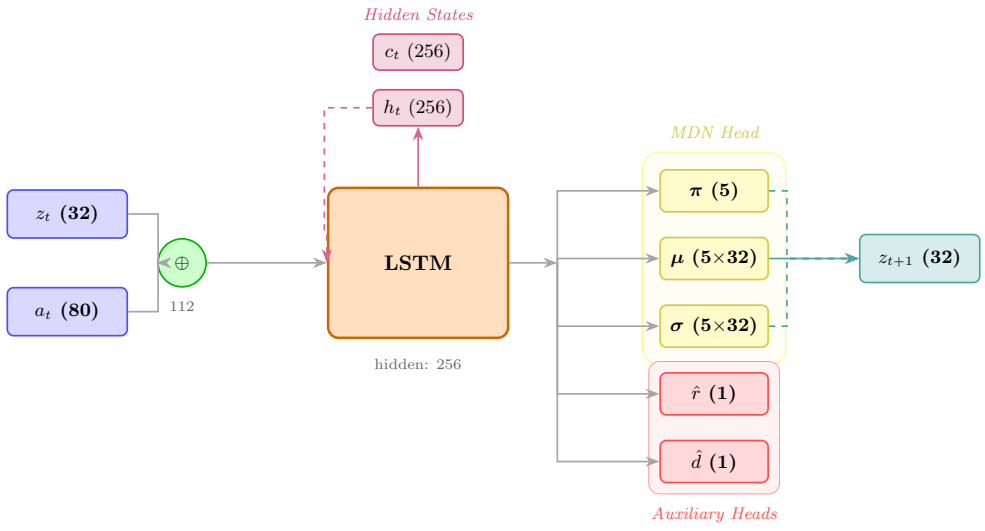

Figure 7: MDN-RNN Architecture. The latent state z_t (32-dim) and action a_t (80-dim) are concatenated and processed by an LSTM (256 hidden units). The MDN head outputs mixture weights $\boldsymbol{\pi}$ (5 components), means $\boldsymbol{\mu}$, and standard deviations $\boldsymbol{\sigma}$ for next-state prediction. Auxiliary heads predict reward $\hat{r}$ and episode termination $\hat{d}$. Hidden states h_t and c_t maintain temporal context.

7.4 Training Results

Data collection was completed with 50,000 frames gathered using a random policy over 250 episodes. VAE training ran for 100 epochs.

Initial Results with MSE Loss: Our first VAE implementation used Mean Squared Error for reconstruction, which produced unacceptably poor results. Reconstructed frames appeared blurry with washed-out colors, failing to preserve sharp entity boundaries. The MSE-trained VAE converged to predicting "safe" averaged pixel values rather than the distinct color regions characteristic of our game (see discussion in Section 7.2).

Improved Results with BCE Loss: After switching to Binary Cross-Entropy loss, reconstruction quality improved dramatically. Figure 8 shows representative results demonstrating that the BCE-trained VAE accurately captures game state including agent position (green), enemy locations (red), and prize positions (yellow). The reconstruction MSE dropped from approximately 0.015 (with MSE loss) to under 0.001 (with BCE loss)—a 15x improvement.

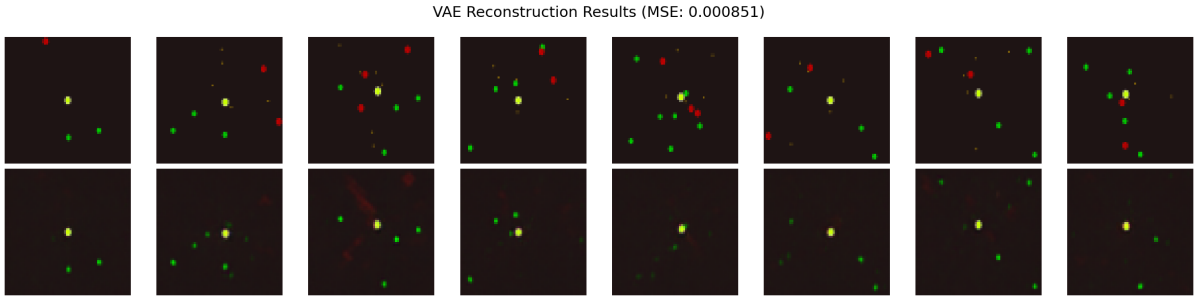

Figure 8: VAE Reconstruction Results with BCE Loss. Top row: original game frames sampled across different game states. Bottom row: reconstructed frames from the 32-dimensional latent encoding. The VAE successfully preserves entity positions, colors, and sharp boundaries. Reconstruction MSE: 0.00085.

7.5 Latent Space Properties

A well-structured latent space enables meaningful interpolation between game states. Figure 9 demonstrates this by linearly interpolating between two latent vectors (z_{start} and z_{end}) with coefficient $\alpha \in [0, 1]$:

$$z_\alpha = (1 - \alpha) \cdot z_{\text{start}} + \alpha \cdot z_{\text{end}} \tag{7}$$

The decoded intermediate frames show smooth semantic transitions: entities gradually shift positions rather than abruptly appearing or disappearing. This geometric meaningfulness is essential for the MDN-RNN to learn coherent dynamics—if nearby latent points represented radically different game states, transition prediction would be impossible.

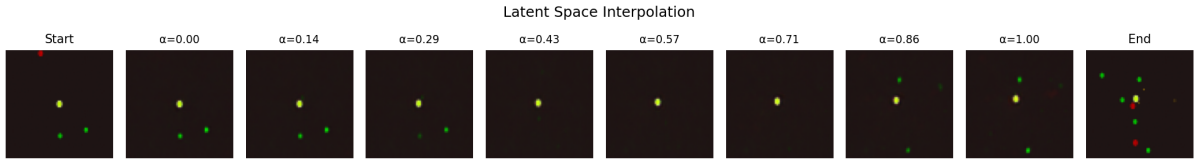

Figure 9: Latent Space Interpolation (BCE-trained VAE). Linear interpolation between two game states produces semantically meaningful intermediate frames. Start and End show actual game states; intermediate images ($\alpha = 0.14$ to $\alpha = 0.86$) are decoded from interpolated latent vectors. Smooth transitions indicate a well-structured latent manifold suitable for dynamics modeling.

7.6 Comparison to DreamerV2

Our world model implementation shares the core VAE + dynamics structure with the original World Models paper but differs from DreamerV2 in several ways:

Table 4: Architectural Comparison between our implementation and DreamerV2. Our staged approach trades end-to-end optimization for clearer debugging and component validation.

Component	Our Implementation	DreamerV2
Latent Type	Gaussian (32-dim)	Categorical (32×32)
Dynamics	MDN-RNN (5 mixtures)	GRU + RSSM
Controller	PPO (SB3)	Actor-Critic in latent space
Training	Staged (VAE $\rightarrow$ RNN $\rightarrow$ PPO)	End-to-end

DreamerV2's end-to-end training and categorical representations offer advantages for complex environments, while our staged approach provides clearer debugging and component validation—appropriate for our 2D game domain.

7.7 MDN-RNN Training Challenges: Reward Prediction and Sigma Collapse

Training the MDN-RNN revealed significant challenges in predicting rewards from latent dynamics. While the model successfully learned latent transition dynamics (MSE < 0.002), the auxiliary reward prediction head failed almost entirely, achieving only 0.08 correlation between predicted and actual rewards.

7.7.1 The Sigma Hacking Problem

In Mixture Density Networks, each Gaussian component outputs a mean μ_k and standard deviation σ_k. The model can "cheat" the negative log-likelihood objective through a phenomenon known as *sigma hacking* or *sigma collapse*. When the model encounters difficult-to-predict samples, it can reduce loss by shrinking σ toward zero for one component and placing μ exactly on the target:

$$-\log p(z|\mu, \sigma) = \frac{(z-\mu)^2}{2\sigma^2} + \log \sigma + \text{const} \tag{8}$$

As $\sigma \to 0$ with $(z-\mu) \to 0$, the loss approaches $-\infty$. To prevent numerical instability, we clamp sigma: $\sigma = \text{clamp}(\exp(\log \sigma), 10^{-6}, 10)$. However, this creates a degenerate solution where sigma stays at the lower bound, making the model overconfident about incorrect predictions on new data.

Diagnosis: Monitoring sigma statistics during evaluation revealed:

- Min sigma: 1.2×10^{-3} (above emergency floor, but very low)
- Mean sigma: 0.077 (indicating tight, potentially overconfident predictions)
- Latent prediction MSE: 0.0016 (good)

These values suggest the model learned latent dynamics reasonably well without full collapse, but the tight sigma indicates limited uncertainty modeling.

7.7.2 Reward Prediction Failure

The more severe problem was reward prediction. Verification revealed:

- Predicted reward range: $[-0.017, +0.034]$ (essentially zero)
- Actual reward range: $[-0.22, +1.30]$
- Correlation: 0.08 (effectively random)

This failure has multiple causes:

Sparse Reward Signal: Game rewards are inherently sparse—positive rewards only occur when hitting enemies or collecting prizes, which happens infrequently in random policy data. The MDN-RNN's reward head, trained with MSE loss, converges to predicting near-zero (the mean reward), achieving low average loss but providing no useful signal for planning.

Random Policy Data Collection: Using random actions for data collection fails to capture reward-rich transitions. An agent randomly moving and shooting rarely hits enemies or collects prizes. The training data therefore contains few examples of positive rewards, starving the reward prediction head.

Loss Function Imbalance: The MDN loss for latent prediction dominates the reward MSE loss (different scales), causing the optimizer to focus primarily on latent dynamics.

7.8 Approach 1: Initial Pure Dream Training (Failed)

7.9 Training Approaches and Results

We evaluated three distinct training strategies for the world model agent. Table 5 summarizes the results across all approaches.

Table 5: Summary of World Model Training Approaches

Approach	Methodology	Training Reward	Real Evaluation
1. Pure Dream	Train on MDN predicted rewards	≈ 0	-5.2 ± 3.1 (Fail)
2. Hybrid	Real rewards + VAE observations	-2.9 ± 3.9	-2.9 (Poor)
3. Dream Fix	Trained policy data + norm rewards	$\mathbf{+74.3 \pm 0.5}$	-6.3 ± 0.1 (Exploit)
4. Dyna-Style	Interleaved real/dream + real eval	-4.9 ± 1.0	$\mathbf{-4.9}$ (Best WM)

1. Pure Dream Training: Our initial attempt followed the original World Models paper, training entirely in imagination. This failed because the random policy data provided sparse rewards, leading the MDN-RNN to predict near-zero rewards everywhere. The agent received no gradient signal and learned a random policy.

2. Hybrid Method (Workaround): We attempted to bypass the broken reward predictor by using real environment rewards while still observing the VAE's latent state. Performance improved slightly (-2.9) but remained poor compared to model-free baselines ($+38.7$). This highlighted a key issue: the 32-dimensional latent space combined with a noisy, uninformative hidden state (trained on random data) lacks sufficient state information for high-performance control.

3. Dream Exploitation: To fix the reward prediction, we collected data using a trained policy and normalized rewards. While this yielded excellent *imagined* performance ($+74.3$), the real-world performance collapsed to -6.3. This revealed **Dream Exploitation**: the agent learned to exploit inaccuracies in the world model rather than learning robust skills.

4. Dyna-Style Training (Best): Addressing these failures, we implemented a Dyna-style approach [12], training with real rewards and periodically evaluating in the real environment. This proved the most stable method, achieving a best real reward of -4.9. While still inferior

to pure model-free RL due to the information bottleneck of the visual latent space, it successfully grounded the agent in reality. While Dyna-style training prevents dream exploitation, the fundamental limitation of the latent observation space remains. The agent still struggles because the 288-dimensional $[z_t, h_t]$ observation lacks the precise structured information available to model-free agents.

7.10 Comprehensive Comparison

Figure 10 compares all world model training approaches against the model-free PPO baseline.

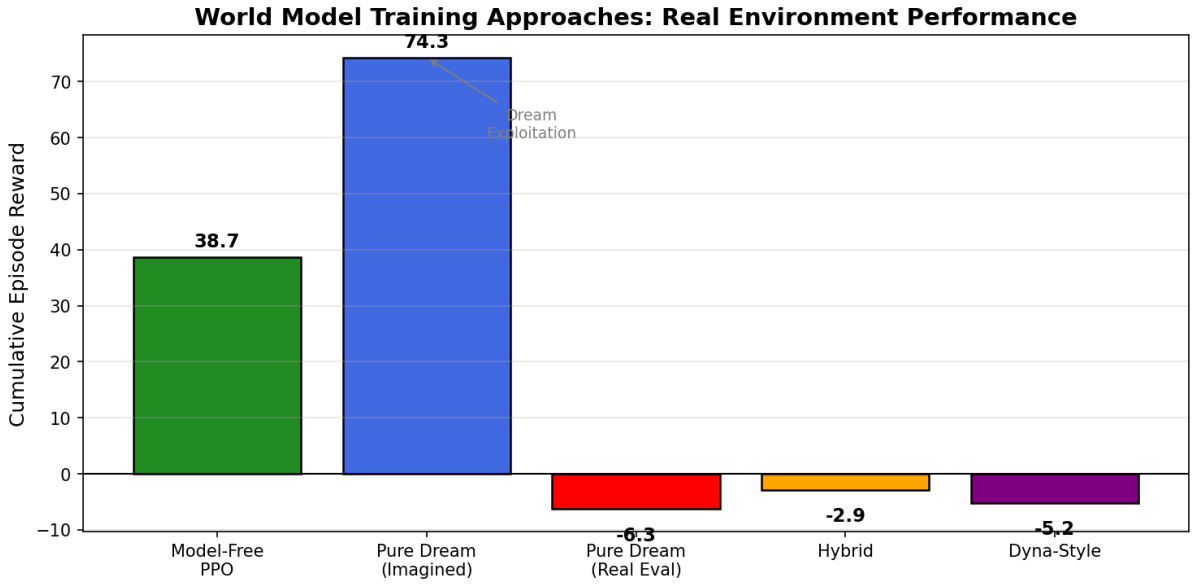

Figure 10: Comparison of world model training approaches. Pure Dream achieves +74.3 in imagination but **catastrophically fails** in real evaluation (-6.3), demonstrating dream exploitation. Hybrid and Dyna-style approaches produce more modest but consistent results. Model-free PPO remains the best performer (+38.74) because it observes structured game state directly.

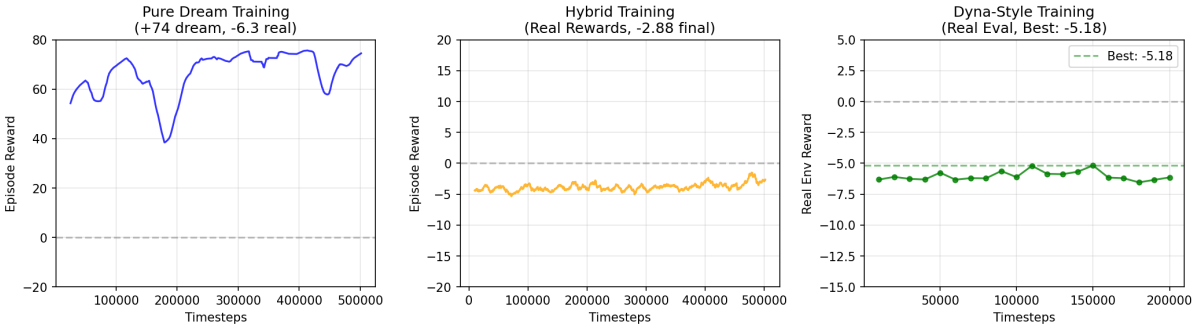

Figure 11: Learning curves for each world model approach. Left: Pure dream training shows rapid reward increase to +74 (in imagination). Center: Hybrid training converges to approximately -2.88 using real environment rewards. Right: Dyna-style periodic real evaluation shows consistent performance around -5 to -6.

7.11 Discussion: Why World Models Underperformed

Our world model experiments revealed fundamental challenges when applying imagination-based training to environments with available structured state:

1. Information Bottleneck: The VAE compresses $64 \times 64 \times 3$ images to a 32-dimensional latent code. While reconstructions appear visually accurate, precise numerical information (exact positions, health values) is lost. Model-free agents observe structured 32-dimensional state with exact values.

2. MDN-RNN Hidden State Utility: The 256-dimensional hidden state, trained on random policy data, provides minimal useful temporal context. Effectively, 256 of the agent's 288 observation dimensions contribute noise rather than signal.

3. Dream Exploitation: When training purely in imagination, the agent discovers action sequences that maximize MDN-RNN predicted rewards but do not correspond to beneficial real-world behaviors. This represents a form of adversarial exploitation of model inaccuracies.

4. Environment Suitability: World models excel when (a) only pixels are available, (b) sample efficiency is critical, or (c) environment dynamics are complex enough to benefit from learned predictive models. Our environment provides structured state and is simple enough that model-free methods are more effective.

Key Insight: The World Models architecture [3] was designed for environments where visual observations are the primary input. Applying it to environments with structured state available introduces unnecessary information loss. This finding is consistent with DreamerV3's success on Atari/DMC (visual only) versus our relatively simple shooter game.

8 Analysis of SAC Performance Failure

Soft Actor-Critic (SAC) significantly underperformed compared to DQN and PPO across all experimental configurations, achieving a best result of only +2.23 on aggressive/medium compared to PPO's +38.74 on aggressive/long—a 17x performance gap. This section analyzes the root causes of SAC's failure in our environment.

8.1 Action Space Mismatch

SAC is designed for continuous action spaces where it can leverage reparameterization gradients through the policy network. Our environment uses a multi-discrete action space (80 combinations), requiring a continuous-to-discrete wrapper that maps box-constrained continuous outputs back to discrete choices. This wrapper introduces quantization noise and prevents direct gradient flow, fundamentally compromising SAC's optimization mechanism.

8.2 Training Speed Penalty

SAC trained at approximately 65 frames per second compared to DQN's 800 FPS and PPO's 1500 FPS aggregate throughput. This 12x speed disadvantage meant SAC experienced far fewer environment interactions per wall-clock hour. Given identical timestep budgets, SAC had less time for policy refinement, particularly problematic for the long (1.6M step) experiments.

8.3 Entropy Regularization Interference

SAC's maximum entropy objective encourages exploration by adding entropy bonuses to the reward. In our relatively simple environment where exploitation of learned behaviors should dominate, the entropy term may have prevented SAC from committing to high-reward action patterns, keeping the policy overly stochastic.

8.4 Episode Length Observations

Table 3 shows that SAC completed significantly more episodes (7,714 for baseline/long vs 2,997 for DQN). This suggests SAC agents died more frequently, accumulating death penalties that

dragged down mean rewards. The shorter average episode length indicates SAC failed to learn survival behaviors as effectively as the other algorithms.

9 Challenges and Issues Encountered

Throughout this project, we encountered several technical challenges that required careful engineering solutions.

Action Space Incompatibility: The multi-discrete action space posed compatibility issues with algorithms designed for discrete (DQN) or continuous (SAC) spaces. We implemented custom wrappers that flatten multi-discrete to discrete for DQN and convert continuous box actions back to discrete choices for SAC.

SAC Training Speed: SAC's off-policy nature with continuous action processing resulted in approximately 12x slower training compared to DQN and PPO. This limited the practical number of experiments we could run with SAC and may contribute to its underperformance.

RGB Rendering for World Model: The arcade-based game environment required custom implementation for rendering to numpy arrays suitable for neural network training, as the default rendering pipeline was designed for display rather than data collection.

10 Conclusions and Next Steps

Our experimental results demonstrate that reward shaping significantly influences RL agent performance in the 2D shooter environment. The aggressive reward configuration consistently outperforms alternatives across all algorithms and training durations. Among algorithms, PPO and DQN achieve comparable performance, while SAC underperforms due to action space conversion overhead and slower training.

Key findings include:

- Reward design dominates algorithm choice in determining final performance
- High standard deviations reflect environmental stochasticity, not training instability
- DQN's reward decrease in sparse-reward scenarios highlights value-based methods' brittleness
- World model approach with BCE loss successfully learns latent representations

Future work will focus on completing world model training and comparing imagination-based planning with model-free baselines. We also plan to investigate hyperparameter sensitivity and conduct longer training runs for the best-performing configurations.

References

[1] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[2] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[3] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[4] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

[5] Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2021.

[6] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[7] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Stephanie Gao, Hunter Henry, Marwan Mattar, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[8] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[9] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[12] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[13] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[14] Tristan Tomilin, Meng Fang, Yudi Zhang, and Mykola Pechenizkiy. Coom: A game benchmark for continual reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.