INF367A 25V / Selected Topics in Artificial Intelligence II

Jørgen Mjaaseth*[1,2] and Morten Blørstad†[1,3]

[1]Department of Informatics, University of Bergen
[2]Adepth Minerals AS
[3]Visito AS
{jmj071, morten.blorstad}@uib.no

Abstract

In this project, we compete in the NeurIPS 2024 competition "Lux AI Season 3" (LuxAI), a reinforcement learning challenge hosted on Kaggle. The competition environment presents several challenges, such as high complexity, partial observability, and handling multiple units. To address these challenges, our approach incorporates two novel methods from recent research in reinforcement learning and planning: Latent Dynamic Robust Representations for World Models[1] and Deep Hierarchical Planning from Pixels[2]. These state-of-the-art techniques aim to tackle the game's complexity by improving representation learning and hierarchical decision-making under partial observability. Due to the significant challenges in integrating these components into a unified multi-agent system, the empirical results did not demonstrate strong performance, but we believe the approach is a promising direction for scalable and robust reinforcement learning.

1 Introduction

Reinforcement Learning (RL) has achieved remarkable progress in recent years, particularly in domains where agents must learn complex behaviors through interaction with dynamic environments. However, challenges remain in sample efficiency, generalization, long-horizon planning and tackling sparse rewards. One promising direction to address these challenges involves the use of *world models*, which allow agents to learn internal representations of their environment's dynamics, enabling them to plan and simulate future outcomes without requiring constant interaction with the real environment.

World models were popularized by Ha and Schmidhuber [3], who proposed a compact generative model that captures the environment's latent dynamics and enables agents to learn policies in a simulated latent space. Building on this, Hafner et al. [4] introduced Dreamer, a model-based RL framework that learns latent dynamics using a *recurrent state-space* (RSSM) model and trains policies entirely within this learned imagination. These approaches demonstrate that learning and leveraging internal environment models can lead to more efficient and scalable RL.

Parallel to this, *hierarchical reinforcement learning* (HRL) offers another avenue for improving RL performance by structuring policies into multiple levels of abstraction. In HRL, high-level policies operate over extended time horizons, setting goals or subgoals that guide low-level controllers. This abstraction can significantly reduce the complexity of the learning problem and enable more efficient exploration. Seminal work by Barto and Mahadevan [5] laid foundational principles for HRL, while more recent work such as Nachum et al. [6] proposed data-efficient hierarchical frameworks for continuous control tasks.

In this report, we investigate the potential of combining world models with hierarchical reinforcement learning. We hypothesize that a hierarchical policy structure operating within a learned world model can leverage the strengths of both paradigms: high-level planning in imagination and fine-grained control at the sensory level. Our goal is to evaluate this combination in a simulated environment with sparse rewards and assess its impact on learning efficiency, performance, and generalization.

2 Background

In this section, we present the background on our solution methods, starting with the world model, followed by the hierarchical RL approach.

2.1 World Models

World models are key in model-based reinforcement learning (MBRL). The Dreamer series (Hafner et al. [4, 7]) is arguably the most successful MBRL algorithm that combines a world model with an actor-critic agent, using an RSSM as a world model to learn latent dynamics. The world model generates imaginary trajectories to train the critic to estimate values and the actor to choose high-return actions. However, as noted by Sun et al. [1], RSSM can overfit to task-irrelevant details, reducing performance. To address this, the authors propose Hybrid-RSSM (HRSSM), a more robust world model architecture.

*Equal Contribution and Corresponding Author.
†Equal Contribution.

Final report for INF367A 25V / Selected Topics in Artificial Intelligence II course, UIB, 2025.

The following sections explain the key components in RSSM and the improvements introduced in HRSSM.

2.1.1 RSSM

The RSSM world model in Dreamer consists of the following components (see Figure A.1 and A.2):

$$
\begin{aligned}
\text{Encoder:} \quad & z_t \sim q_\phi(z_t \mid h_t, o_t) \\
\text{Decoder:} \quad & \hat{o}_t \sim p_\phi(\hat{o}_t \mid h_t, z_t) \\
\text{Recurrent model:} \quad & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\
\text{Transition predictor:} \quad & \hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t) \\
\text{Reward predictor:} \quad & \hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t) \\
\text{Continue predictor:} \quad & \hat{c}_t \sim p_\phi(\hat{c}_t \mid h_t, z_t).
\end{aligned} \tag{1}
$$

The encoder maps an observation o_t into a stochastic state z_t. A recurrent model updates the hidden state h_t, which summarises temporal dependencies. A transition predictor predicts future stochastic states $\hat{z}_t$ without access to observations. The decoder reconstructs the input $\hat{o}_t$ to ensure informative representations. Additionally, a reward predictor and a continue predictor are trained to model task-relevant outcomes such as rewards and episode terminations. The latent state is given by $s_t = [h_t, z_t]$, which forms the basis for all predictions. We can do imagination-based learning by using the world model to roll out imaginary trajectories and train the actor and critic (see Figure A.2 for more details).

2.1.2 Improving Robustness

To improve robustness and improve task-specific representations, HRSSM introduces 1) *dual-branch architecture* with a raw and masked branch, 2) a *spatio-temporal masking strategy* that masks patches across space and time in the masked branch to make it more robust to irrelevant input features, (3) a *latent reconstruction* loss to align the masked and raw branches, and (4) a *bisimulation* objective to capture task-relevant information by ensuring observations with similar reward are closer in the latent space (see Figure 1). The masked branch of HRSSM consists of the same components as RSSM (Equation 1), whereas the raw branch only consists of the recurrent, transition and encoder model (see Table A.1). The masked branch is optimised with stochastic gradient descent, using the losses described in Appendix B.1. The raw branch is updated using an exponential moving average of the masked branch parameters.

2.2 Hierarchical Approach

The paper *Deep Hierarchical Planning from Pixels* (Hafner et al. [2]) proposes a framework that integrates hierarchical planning with representation learning directly from pixel observations. The method, called **Director**, is a HRL framework designed to learn long-horizon behaviors directly from pixel inputs. It integrates a learned world model with a two-tier policy structure: a high-level *manager* and a low-level *worker*. The manager selects abstract goals in a discrete latent space, while the worker executes primitive actions to achieve these goals. This hierarchical decomposition enables efficient exploration and planning in complex environments with sparse rewards.

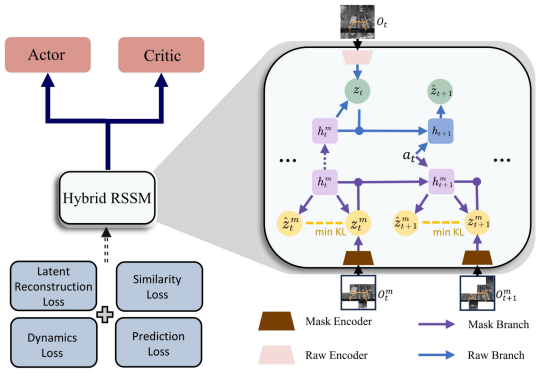


Figure 1. Hybrid-RSSM and actor-critic architecture. The Hybrid-RSSM learns robust and task-relevant representations using four objectives: (1) latent reconstruction to align features from masked and unmasked observations, (2) a bisimulation-based similarity loss to preserve behavioral equivalence, and (3–4) two standard objectives from Dreamer (Hafner et al., 2020; 2023) for learning latent dynamics and predicting rewards and continue flags.

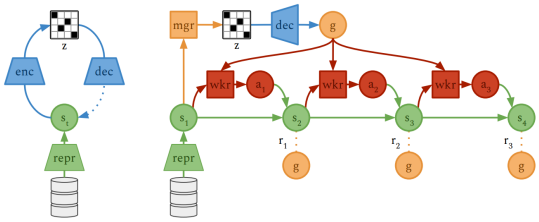


Figure 2. Overview of the Director framework. Left: The Goal Variational Autoencoder. Right: Visualization of goal / action selection process and temporal abstraction.

Goal Autoencoder To facilitate goal selection in a manageable space, Director employs a goal autoencoder that compresses high-dimensional latent states $s_t \in \mathbb{R}^{1024}$ from the world model into discrete codes z. The encoder maps s_t to a sequence of categorical variables z, and the decoder reconstructs the latent state $g = \text{dec}(z)$. This discrete representation simplifies the manager's goal selection process and ensures that selected goals correspond to plausible states.

Manager Policy Operating at a coarser temporal resolution, the manager policy $\pi_{\text{mgr}}(z|s_t)$ selects a goal code z every K steps (e.g., $K = 8$). The selected code is decoded to a latent goal $g = \text{dec}(z)$, which the worker aims to achieve. The manager's objective is to maximize a combination of extrin-

sic task rewards and an intrinsic exploration bonus. The exploration bonus is derived from the reconstruction error of the goal autoencoder, encouraging the agent to explore novel states.

Worker Policy The worker policy $\pi_{\text{wkr}}(a_t|s_t, g)$ operates at every time step, receiving the current state s_t and the goal g from the manager. Its objective is to produce actions that drive the agent towards achieving the goal g. The worker is trained using imagined rollouts from the world model, focusing solely on goal achievement without direct access to task rewards. See Appendix B.2 for details on optimization and loss functions.

3 Competition & Environment

The "Lux AI Season 3" [8] competition, part of NeurIPS 2024 and hosted by Kaggle, tasks participants with designing intelligent agents that operate in a challenging, partially observable, two-player environment. The game unfolds across a 24x24 procedurally generated 2D grid. Each game consists of a best-of-five match sequence, where each match spans 100 time steps. Both teams control units tasked with exploring the map, collecting relic points, and denying their opponent's progress.

The game environment includes several tile types: Empty, Asteroid (impassable), Nebula (which may reduce vision and sap energy), Energy Nodes (which emit harvestable energy fields), and Relic Nodes (which award points when units occupy special hidden tiles nearby). Asteroids and other map elements may shift over time, and relic scoring zones are discoverable only through active exploration.

Fog of war adds another layer of complexity. Each unit contributes to the team's vision using a distance-based sensor model. This is visualized in figure 3 as pink tiles. Nebula tiles can diminish this vision, further challenging decision-making under uncertainty. Additionally, agents must manage energy levels—depleting energy prevents action and units must recharge near energy fields or risk being removed through opponent sap attacks.

The Lux AI environment presents a dynamic, multi-agent, partially observable challenge with sparse rewards, complicating RL experimentation. Agents must adapt over long horizons, with delayed action effects making credit assignment difficult. Success likely hinges on learning compact representations, managing uncertainty, and employing hierarchical planning.

4 Method

In this section, we describe the two primary approaches explored in our experiments: a baseline method leveraging standard RL techniques, and a novel architecture integrating hierarchical reasoning and a learned world model.

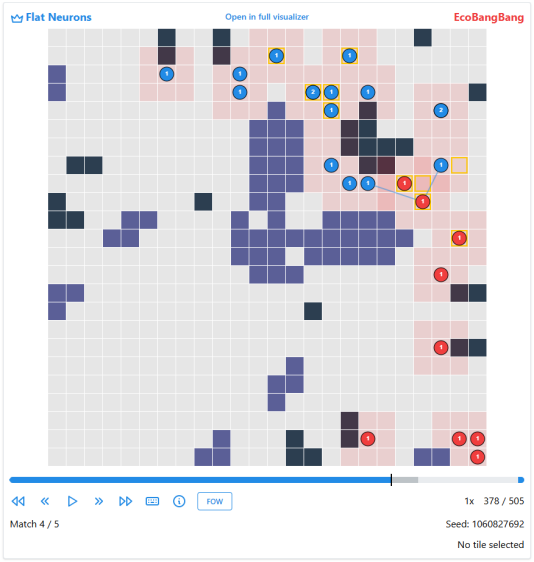


Figure 3. Visualization of a Lux AI game between teams 'Flat Neurons' (#1) and 'EcoBangBang' (#6). Blue circle = Player 1 ship. Red circle = Player 2 ship. Black tile = Asteroid. Purple tile = Nebula. Pink tile = Visible tile. Tiles outlined in yellow are relic tiles. Blue line represents a zap (attack).

For the baseline, we implemented Proximal Policy Optimization (PPO), which uses the environment's raw extrinsic rewards to learn a policy. The goal is to train PPO to select actions for all ships simultaneously at each timestep. Although we aimed to keep the baseline as simple as possible, we introduced our *Universe* class—described later in this chapter—to process the environment's observations. This addition ensures a consistent interface and allows for a more meaningful comparison with our proposed method, which relies heavily on the same abstraction.

Our proposed method builds upon this setup by introducing a multi-agent capable Director architecture, powered by a Hybrid-RSSM. Like the baseline, observations are first processed through the *Universe*. The output is then consumed by the world model, which learns a latent representation of the environment and enables the system to imagine future trajectories. At the top level, a multi-agent Manager assigns individual high-level goals to each ship. At the lower level, a multi-agent Worker translates those goals into concrete actions. This implements a multi-agent extension of *Director*, shown in Figure 2, where the "repr" module corresponds to the Hybrid-RSSM.

Both components are implemented using our custom multi-agent version of Proximal Policy Optimization (PPO).

The specific architectural components and training procedures underlying this method are detailed

in the sections that follow.

4.1 Capturing Information

Before training begins, a significant portion of our effort goes into capturing and modeling the environment to enable effective downstream learning. Due to the stochastic and partially observable nature of Lux AI Season 3, it is critical to structure the game's state space into interpretable, predictable components.

In the `Universe` class, we compute a sequence of future universe states $\{u_t, u_{t+1}, \ldots, u_{t+H}\}$ conditioned on the current observation o_t, i.e., $\{u_{t:t+H} \mid o_t\}$. As a first step, the observation space is decomposed into semantically meaningful layers, as outlined in table 1.

Table 1. A subset of the output from the `Universe` class. Several parameters have been omitted for clarity and to maintain the simplicity and readability of the figure.

Symbol	Name	Description	Dimension
u_{P_1}	Player 1 Units	$P(P_1\text{-ship})$, per tile	$\mathbb{N}^{24\times24}$
u_{P_2}	Player 2 Units	$P(P_2\text{-ship})$, per tile	$\mathbb{N}^{24\times24}$
u_O	Observeable Tiles	Tile is observeable	$\mathbb{Z}_2^{24\times24}$
u_A	Asteroid Tiles	Tile is asteroid	$\mathbb{Z}_2^{24\times24}$
u_N	Nebula Tiles	Tile is nebula	$\mathbb{Z}_2^{24\times24}$
u_E	Energy Tiles	Energy level of tile	$\mathbb{Z}^{24\times24}$
u_R	Relic Tiles	Tile is relic	$\mathbb{Z}_2^{24\times24}$

These entries are stacked into a unified representation $S \in \mathbb{Z}^{24\times24\times7}$. For predictive purposes, we consider the horizon of future states $u_{t:t+H} \in \mathbb{Z}^{24\times24\times7\times H+1}$. For simplicity, some properties computed by the `Universe` class have been omitted from this explanation.

Nebula and Asteroid Prediction. The locations of dynamic tile types (nebulae and asteroids) are learned by analyzing historical tile movements. By tracking changes over multiple time steps, we estimate the parameters governing their rate and direction of movement. Once the pattern is inferred, we can roll forward the tiles' positions over future time steps deterministically.

Player Position Forecasting. For each unit, we model future positions as a probability distribution over the grid. This takes into account legal move directions, current energy, and tile constraints (e.g., impassable asteroids). Probabilities for next-step positions are computed and then recursively propagated to build a temporal probability map. The resulting model estimates $P(pos_t)$ and updates it step-by-step for the horizon.

Determining Relic Positions. The Relics class tracks hidden relic tiles in a 24x24 grid. It maintains three key lists: nodes (observed relic positions), empty (confirmed empty tiles), and relics (confirmed relic tiles).

The number of points scored in each timestep can be calculated from information available in an observation. Furthermore, if a ship scores points, it must be on a relic tile. The EquationSet class helps solve ambiguous cases by maintaining a system of equations where each equation represents a set of possible relic positions that could explain the points scored. Together, this system builds a probabilistic model of relic locations. The returned probability map is constructed by:

- Tracking confirmed relic and empty tiles
- Using point observations to eliminate possibilities
- Solving equations to determine probabilities for ambiguous cases
- Maintaining symmetry (if a tile is a relic, its symmetric counterpart is too)

The collected output of the `Universe` class captures dynamic, observable, and hidden structures in the environment, which we hope can support better planning and improve sample efficiency during downstream training.

4.1.1 World Model Implementation

The HRSSM implementation is available on GitHub[1]. However, it was not directly compatible with the LuxAI environment and the observation format used in our `Universe` class. To adapt it to our setup, we made two key modifications: (1) implemented a custom sequence replay memory for storing observations, and (2) modified the encoder to match our observation space. With these changes, we successfully integrated HRSSM as the representation model for the hierarchical approach described in the next section. There were also challenges with imagination-based learning, which we needed to adjust as Dreamer and HRSSM are implemented for a single agent, whereas we are dealing with multiple agents. More on this in Section 4.1.2.

4.1.2 Director implementation

The original implementation of Director, as proposed by the authors, is not designed for multi-agent settings. However, Lux AI Season 3 is inherently a multi-agent game, where ships may accumulate experience at different rates—some may remain inactive for long periods. To address this, we extended Director by implementing a multi-agent Proximal Policy Optimization (PPO) system. To learn effectively in a multi-agent environment, this system is structured around the following key architectural components:

- A shared actor-critic network (`CommonAC`) that all agents (workers or managers) learn from
- Individual agent (`BehaviourAC`) that interacts with the environment
- A centralized training mechanism where experiences from all agents are used to update the common policy

[1] https://github.com/bit1029public/HRSSM

- Support for both continuous and discrete action spaces
- A buffer system to store and process experiences from multiple agents

This architecture allows agents to benefit from shared learning while still maintaining distinct trajectories and action contexts. It enables scalable training across multiple agents with varying levels of activity and exposure.

5 Results

We trained both the baseline PPO and the director agent for 20 games against the rule-based agent that followed the environment (See Appendix D)[2]. The training losses and the agents' scores per match across the games can be found in Appendix C.1 and C.2. After training, we conducted an evaluation tournament to compute each agent's ELO rating. Table 2 summarises the tournament results.

The evaluation tournament shows that none of the RL agents were able to outperform the rule-based agent. Nonetheless, on a positive note, our improvement ideas from the papers selected improved the performance of the agent, with the Director agent achieving a higher ELO rating than the PPO agent. Based on the evaluation, the rule-based agent would be agent selected as our final agent.

Agent	ELO Rating
Rule-based	**1110**
Director	969
PPO	921

Table 2. ELO ratings after the evaluation tournament. Each agent plays every other agent in 5 games, starting with an initial rating of 1000. After each game, the ELO ratings are updated based on the game outcomes. Agent in **Bold** highlights the best agent which we would have selected for the Lux competition.

6 Discussion and Conclusion

In this work, we aimed to develop a novel agent architecture by integrating ideas from hierarchical RL and HRSSM. While we included a PPO baseline for comparative purposes, we emphasize that the majority of our effort was directed toward designing and implementing the more complex Director/HRSSM system. The PPO baseline serves primarily to establish a reference point rather than to represent a major contribution.

The complexity of the main architecture stems from the integration of multiple state-of-the-art methods, each introducing distinct loss functions, training regimes, and latent representations. Combining these into a unified system was further complicated by the challenges inherent in multiplayer, partially observable settings. In practice, aligning the training processes and ensuring consistent latent representations across components proved to be a substantial undertaking.

In an attempt to facilitate learning, we experimented with several reward shaping techniques, utilizing our Universe abstraction class. Despite these efforts, reward shaping did not yield significant improvements. Given additional time, we believe that further leveraging the Universe abstraction to gather and process more detailed environmental information could enhance learning and performance.

In terms of implementation, we closely followed the loss function descriptions provided in the original papers. However, ambiguities and omissions in the mathematical formulations introduced difficulties for the losses in Hafner et al. [2]. Notably, the absence of techniques such as loss clipping, which we did not incorporate, likely contributed to the high variance and magnitude of observed loss values, as shown in Figure C.3(b) and C.3(c).

Director utilizes extrinsic rewards from the environment to optimize the manager policy. However, in multi-agent settings where multiple managers independently select goals, the lack of clear attribution between individual goals and resulting rewards becomes a significant challenge. This problem of credit assignment undermines effective learning. A promising solution may involve further development of our *Relics* abstraction class, which could offer a system-specific and effective approach to resolving this issue. Additionally, it may be worthwhile to explore techniques from the broader field of multi-agent reinforcement learning; for instance, Counterfactual Multi-Agent Policy Gradients (COMA)[9] could provide a principled mechanism for addressing credit assignment among managers.

Although the empirical results obtained in this work did not demonstrate strong performance, they provide valuable insights into the practical challenges of integrating advanced RL frameworks within complex multi-agent environments. Despite the difficulties encountered, the architectural approach explored here—combining hierarchical planning with learned world models—remains a promising direction for future research. Further investigation into stabilization techniques, improved credit assignment mechanisms, and more effective information abstraction may help to realize the potential of such systems in rich, partially observable environments.

[2] Earlier training was longer (days) but inconsistent. For fair comparison, we retrained both agents over 20 games.

References

[1] R. Sun, H. Zang, X. Li, and R. Islam. "Learning Latent Dynamic Robust Representations

for World Models". In: *Proceedings of the 41st International Conference on Machine Learning*. Ed. by R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp. Vol. 235. Proceedings of Machine Learning Research. PMLR, 21–27 Jul 2024, pp. 47234–47260. URL: https://proceedings.mlr.press/v235/sun24n.html.

[2] D. Hafner, K.-H. Lee, I. Fischer, and P. Abbeel. *Deep Hierarchical Planning from Pixels*. https://arxiv.org/pdf/2206.04114. 2022. arXiv: 2206.04114 [cs.AI].

[3] D. Ha and J. Schmidhuber. "World Models". In: (2018). DOI: 10.5281/ZENODO.1207631. URL: https://zenodo.org/record/1207631.

[4] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. *Dream to Control: Learning Behaviors by Latent Imagination*. 2020. arXiv: 1912.01603 [cs.LG]. URL: https://arxiv.org/abs/1912.01603.

[5] A. G. Barto and S. Mahadevan. "Recent advances in hierarchical reinforcement learning". In: *Discrete Event Dynamic Systems* 13.1-2 (2003), pp. 41–77.

[6] O. Nachum, S. Gu, H. Lee, and S. Levine. *Data-Efficient Hierarchical Reinforcement Learning*. 2018. arXiv: 1805.08296 [cs.LG]. URL: https://arxiv.org/abs/1805.08296.

[7] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. *Mastering Diverse Domains through World Models*. 2024. arXiv: 2301.04104 [cs.AI]. URL: https://arxiv.org/abs/2301.04104.

[8] S. Tao, A. Kumar, B. Doerschuk-Tiberi, I. Pan, A. Howard, and H. Su. *NeurIPS 2024 - Lux AI Season 3*. https://kaggle.com/competitions/lux-ai-season-3. Kaggle. 2024.

[9] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. *Counterfactual Multi-Agent Policy Gradients*. 2024. arXiv: 1705.08926 [cs.AI]. URL: https://arxiv.org/abs/1705.08926.

[10] T. Stone. *Lux-Design-S3/kits/python/agent.py*. https://github.com/Lux-AI-Challenge/Lux-Design-S3/blob/main/kits/python/agent.py. 2013.

A Word Model

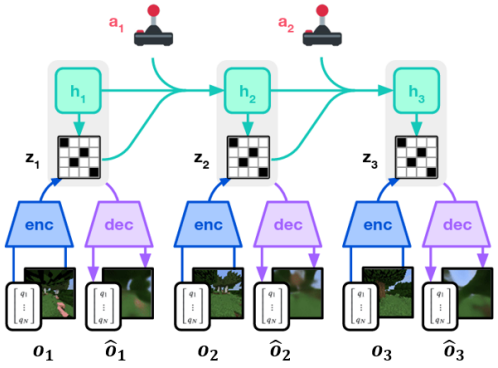


Figure A.1. RSSM from Dreamer (Hafner et al., 2023). From a replay memory of past experience, the world model encodes raw observations into representations z_t that are predicted by a transition model with hidden state h_t given action a_t.

	Mask Branch	Raw Branch
Masker	$o_t^m = \varepsilon_\phi(o_t)$	
Recurrent	$h_t^m = f_\phi(h_{t-1}^m, z_{t-1}^m, a_{t-1})$	$h_t = f'_\phi(h_{t-1}^m, z_{t-1}, a_{t-1})$
Encoder	$z_t^m \sim q_\phi(z_t^m \mid h_t^m, o_t^m)$	$z_t \sim q'_\phi(z_t \mid h_t^m, o_t)$
Transition	$\hat{z}_t^m \sim p_\phi(\hat{z}_t^m \mid h_t^m)$	$\hat{z}_t \sim p'_\phi(\hat{z}_t \mid h_t)$
Reward	$\hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t^m, z_t^m)$	
Continue	$\hat{c}_t \sim p_\phi(\hat{c}_t \mid h_t^m, z_t^m)$	

Table A.1. Comparison of model components in the Mask and Raw branches of HRSSM. Superscript ^{m} denotes masked observations.

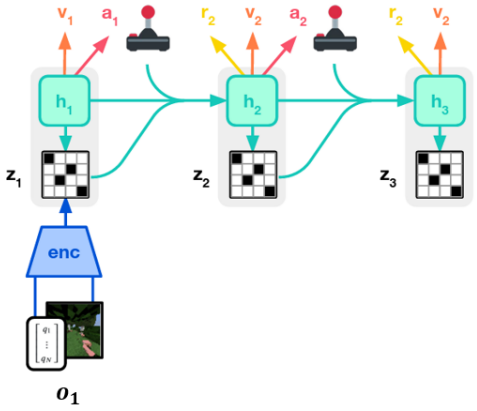


Figure A.2. Imagination-based Learning: The observation o_t is encoded into a latent state z_t, combined with a recurrent hidden state h_t to form the model state $s_t = [h_t, z_t]$. The actor selects actions a_t and the critic estimates values v_t based on s_t. Using the recurrent model and transition predictor, the agent rolls out imagined future states without access to further observations. The actor and critic are trained using these imagined trajectories.

B Losses

B.1 HRSSM Loss Functions

The dynamics loss $\mathcal{L}_{\text{dyn}}(\phi)$ trains the masked branch by aligning the encoder and transition using two KL

terms with free-bits regularisation

$$\begin{aligned}\mathcal{L}_{\text{dyn}}(\phi) &:= \beta_1 \max(1, \mathcal{L}_1(\phi)) + \beta_2 \max(1, \mathcal{L}_2(\phi)) \\ \mathcal{L}_1(\phi) &:= \mathrm{KL}\left[\mathrm{sg}\left(q_\phi(z_t^m \mid h_t^m, e_t^m)\right) \parallel p_\phi(\hat{z}_t^m \mid h_t^m)\right] \\ \mathcal{L}_2(\phi) &:= \mathrm{KL}\left[q_\phi(z_t^m \mid h_t^m, e_t^m) \parallel \mathrm{sg}\left(p_\phi(\hat{z}_t^m \mid h_t^m)\right)\right].\end{aligned} \tag{2}$$

Here, $\mathrm{sg}(\cdot)$ is the stop-gradient. $\mathcal{L}_1$ trains the transition predictor to match the encoder, while $\mathcal{L}_2$ ensures the encoder stays predictable. We use $\beta_1 = 0.5$ and $\beta_2 = 0.1$.

The latent reconstruction loss aligns the masked and raw branches in latent space by minimising the mean squared error (MSE) between their normalised latent representations $\bar{s}_t$ and $\bar{s}_t^m$,

$$\mathcal{L}_{\text{rec}}(\phi) := \mathrm{MSE}(\bar{s}_t, \bar{s}_t^m). \tag{3}$$

The bisimulation loss encourages task-relevant representations by minimising bisimulation error,

$$\begin{aligned}\mathcal{L}_{\text{sim}} &:= \left(d(s_i, s_j^m) - \mathcal{F}^\pi d(s_i, s_j^m)\right)^2 \\ &= \left(d(s_i, s_j^m) - \left(\left|r_{s_i}^\pi - r_{s_j}^\pi\right| + \gamma d(\hat{s}_{i+1}, \hat{s}_{j+1}^m)\right)\right)^2.\end{aligned} \tag{4}$$

The goal is to ensure that states with similar rewards and dynamics are close in latent space. Here, $d(\cdot, \cdot)$ is the cosine distance, and $\mathcal{F}^\pi$ denotes the expected future similarity under policy π.

The prediction loss,

$$\mathcal{L}_{\text{pred}}(\phi) := -\ln p_\phi(r_t \mid s_t^m) - \ln p_\phi(c_t \mid s_t^m), \tag{5}$$

consists of two terms: the symlog[3] reward prediction loss and the binary classification loss for episode continuation. Both are computed from the masked branch latent state s_t^m.

B.2 Director Loss Functions

All components are trained concurrently using gradients computed from imagined trajectories generated by the world model, allowing for efficient and scalable learning.

Goal Autoencoder The world model representations $s_t \in \mathbb{R}^{1024}$ are high-dimensional, making direct goal selection difficult for the manager. To reduce this complexity, *Director* uses a goal autoencoder to compress s_t into discrete codes z. It consists of a Goal Encoder $\mathrm{enc}_\phi(z \mid s_t)$ and a Goal Decoder $\mathrm{dec}_\phi(z) \approx s_t$ (3). Rather than assigning each state a unique category, which limits generalization, Director adopts a factorized categorical representation as in DreamerV2: the encoder outputs an 8×8 matrix of logits, samples one-hot vectors from each row, and flattens them into a sparse vector. Training uses straight-through estimation, optimizing:

$$\mathcal{L}(\phi) = \left\|\mathrm{dec}_\phi(z) - s_t\right\|^2 + \beta\, \mathrm{KL}\left(\mathrm{enc}_\phi(z \mid s_t) \,\|\, p(z)\right),$$

where $z \sim \mathrm{enc}_\phi(z \mid s_t)$.

Manager Director learns a manager policy that selects a new goal every $K = 8$ time steps. Instead of operating in the continuous latent space of the world model, the manager outputs abstract actions in the discrete code space of the goal autoencoder. Its policy is defined as:

$$\text{Manager Policy:} \quad \mathrm{mgr}_\psi(z \mid s_t) \tag{5}$$

To encourage temporally-abstract exploration, the manager is trained to maximize the expected sum of task rewards and exploration rewards. The exploration bonus is computed using the reconstruction error of the goal autoencoder, rewarding novel states:

$$r_t^{\text{expl}} = \left\|\mathrm{dec}_\phi(z) - s_{t+1}\right\|^2, \quad z \sim \mathrm{enc}_\phi(z \mid s_{t+1}) \tag{6}$$

Both extrinsic and exploration returns are normalized by their exponential moving standard deviations, then combined using weights $w_{\text{extr}} = 1.0$ and $w_{\text{expl}} = 0.1$. The manager learns two critics for bootstrapping rewards beyond the imagination horizon and for variance reduction.

Worker The worker aims to reach goals $g = \mathrm{dec}(z)$ decoded from discrete codes z chosen by the manager. Conditioning on decoded goals decouples the worker's learning from the goal autoencoder. The worker policy is defined as:

$$\text{Worker Policy:} \quad \mathrm{wkr}_\xi(a_t \mid s_t, g) \tag{7}$$

To encourage goal-reaching behavior, the reward function compares the current state s_{t+1} to goal g using a modified cosine similarity:

$$r_t^{\text{goal}} = \frac{g^\top s_{t+1}}{m^2}, \quad m = \max(\|g\|, \|s_{t+1}\|)$$

This *max-cosine reward* preserves directional alignment while penalizing magnitude mismatch, avoiding artifacts of L2-based rewards. Note that this equation is somewhat ambiguous in its interpretation, leaving room for multiple valid implementations. To address this, we experimented with several variants, and the impact of these differences is reflected in the variability observed in the presented results.

[3] Symlog is a link function used to learn more robust predictions. A detailed description can be found on page subsection "Robust predictions" on p. 7 in Hafner et al. [7].

C Training

C.1 PPO Agent

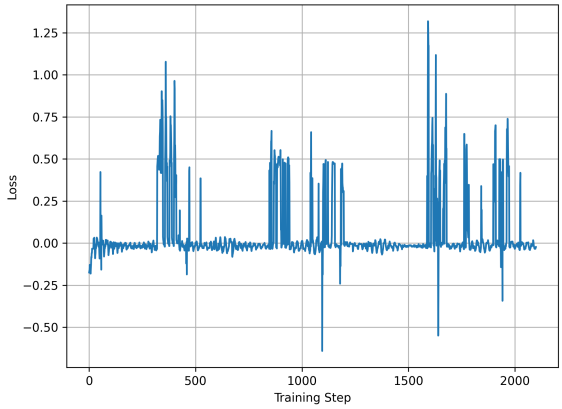


Figure C.1. PPO loss (actor + critic loss) development per training step.

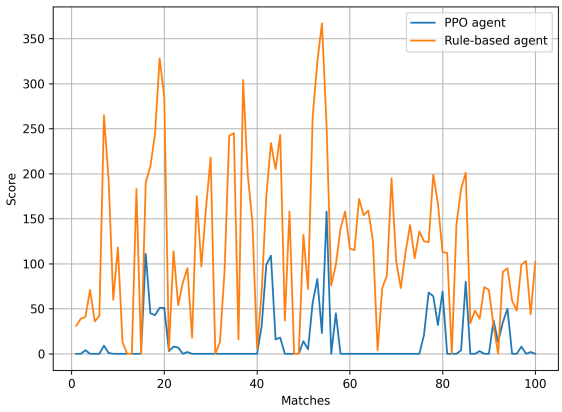
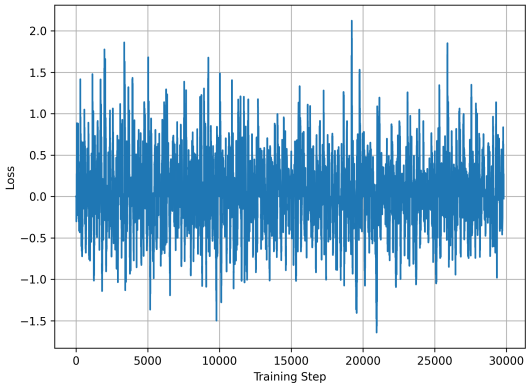
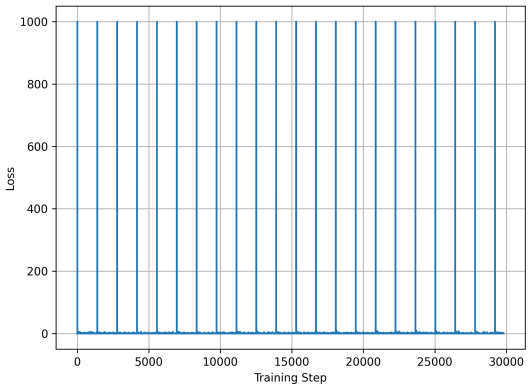


Figure C.2. Score per match comparison: PPO vs. Rule-based agent.

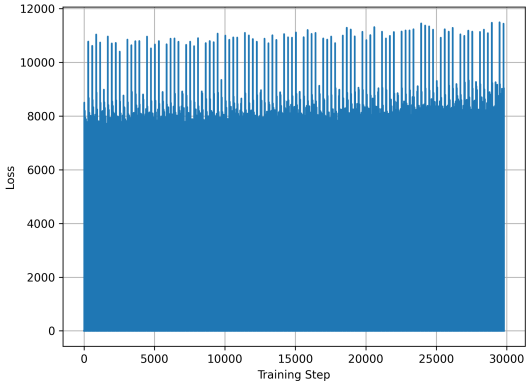
C.2 Our Agent



(a) Worker loss

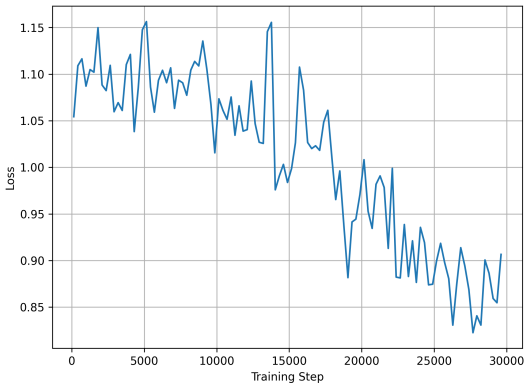


(b) Manager loss

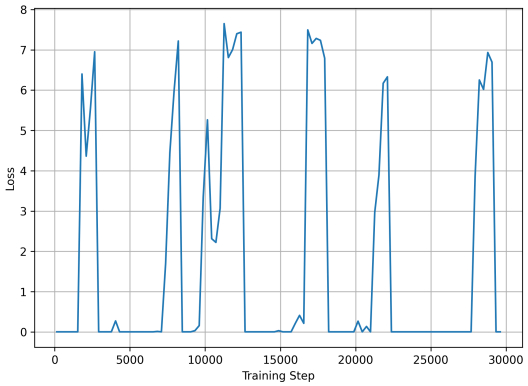


(c) Goal loss

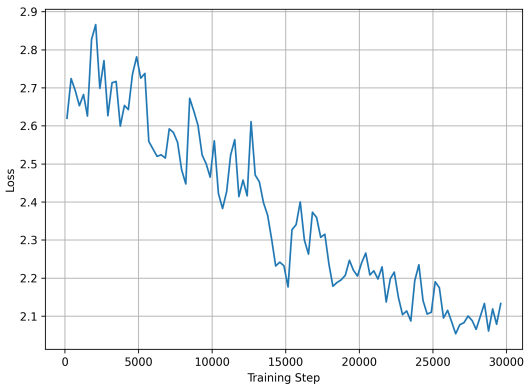
Figure C.3. Training loss curves for different components of the director. The manager's loss values have been clipped at 1000 for visualisation purposes.



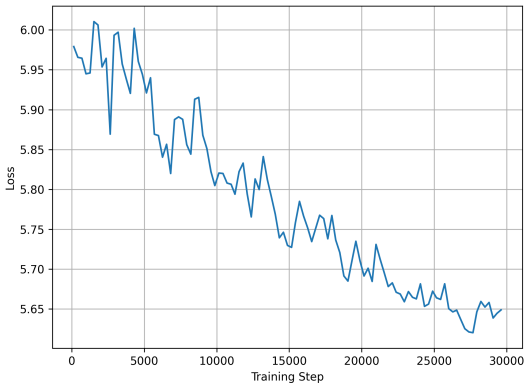
(a) Latent reconstruction loss



(b) Bisimulation loss



(c) Dynamics loss



(d) Prediction loss

Figure C.4. Training loss curves for different components of the world model.

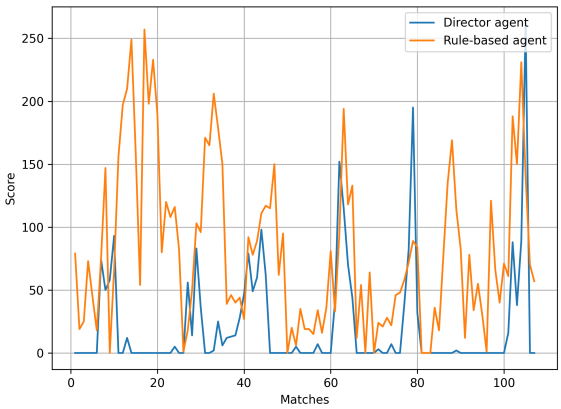


Figure C.5. Score per match comparison: Our vs. Rule-based agent.

D Rule based agent

The GitHub repository for Lux AI Season 3 includes an agent [10], referred to as the 'Rule based agent', that can be used out-of-the-box for gameplay. The agent implements a two-phase strategy for controlling units in the game. Initially, units explore the map randomly by moving to random locations every 20 steps. When a relic node is discovered, the strategy shifts to a relic-focused approach. Units within a Manhattan distance of 4 from the nearest relic node perform random movements to hover around it, while other units move directly towards the relic node. The agent maintains a memory of discovered relic nodes across matches, storing their positions in $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$. The movement direction d for each unit is determined by the function $d = \text{direction\_to}(p_{\text{unit}}, p_{\text{target}})$, where p_{unit} is the unit's position and p_{target} is either a random exploration point or the nearest relic node position.