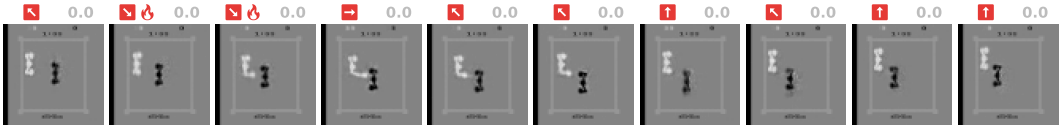
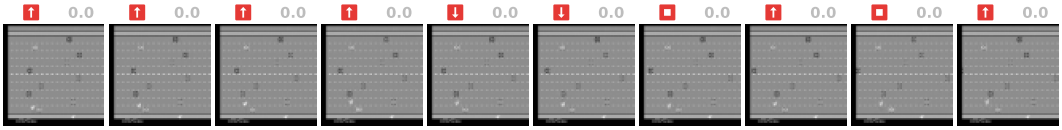
A Appendix

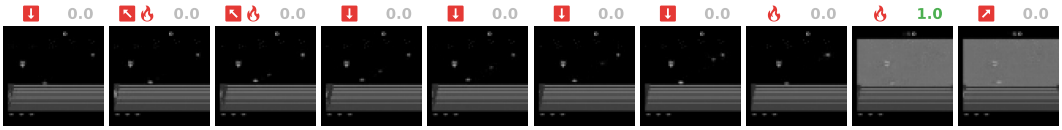
A.1 Extended Experiments



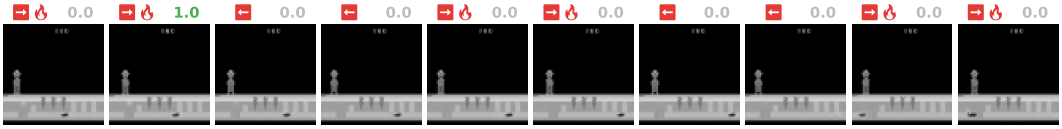
(a) Boxing. The player (white) presses *fire*, misses the opponent, gets no reward, and retreats.



(b) Freeway. The player moves up, steps back because of an approaching car, and continues to move up.



(c) Jamesbond. The player presses *fire* to fire a missile, and gets a reward when it hits.



(d) Gopher. The player (farmer) closes a hole created by the opponent (gopher). The gopher keeps moving independent from the selected actions, indicating that the world model has correctly learned the correlations between the player and the actions. Note that the gopher briefly disappears and reappears on the other side.

Figure 9: Additional example trajectories generated by our world model.

Additional Analysis:

1. We provide more example trajectories in Figure 9.
2. We present more attention plots in Figures 10 and 11. All attention maps are generated using the attention rollout method by Abnar & Zuidema (2020). Note that we had to modify the method slightly, in order to take the causal masks into account.
3. *Sample Efficiency*: We provide the scores of our main experiments after different amounts of interactions with the environment in Table 2. After 50K interactions, our method already has a higher mean normalized score than previous sample-efficient methods. Our mean normalized score is higher than DER, CURL, and SimPLe after 25K interactions. This demonstrates the high sample efficiency of our approach.
4. *Stochasticity*: The stochastic prediction of the next state allows the world model to sample a variety of trajectories, even from the same starting state, as can be seen in Figure 12.
5. *Long Sequence Imagination*: The world model is trained using sequences of length $\ell = 16$, however, it generalizes well to very long trajectories, as shown in Figure 13.
6. *Frame Stacking*: In Figure 14 we visualize the learned stacks of frames. This shows that the world model encodes and predicts the motion of objects.

Additional Ablation Studies:

1. *Thresholded Entropy Loss*: In Figure 15 we compare (i) our thresholded entropy loss for the policy (see Section 2.2) with (ii) the usual entropy penalty. For (i) we use the same hyperparameters as in our main experiments, i.e., $\eta = 0.01$ and $\Gamma = 0.1$. For (ii) we set $\eta = 0.001$ and $\Gamma = 1.0$, which effectively disables the threshold. Without a threshold, the entropy is more likely to either collapse or diverge. When the threshold is used, the score is

higher as well, probably because the entropy is in a more sensible range for the exploration-exploitation trade-off. This cannot be solved by adjusting the penalty coefficient η alone, since it would increase or decrease the entropy in all games.

2. *History Length*: We trained our world model with a shorter history and set $\ell = 4$ instead of $\ell = 16$. This has a negative impact on the score, as can be seen in Figure 16, demonstrating that more time steps into the past are important.
3. *Choice of Policy Input*: In Section 2.2 we explained why the input to the policy is only the latent state, i.e., $x = z$. In Figure 17 we show that using $x = [z, h]$ can result in lower final scores. We hypothesize that the policy network has a hard time keeping up with the changes of the space of h during training and cannot ignore this additional information.
4. *Increasing the Sample Efficiency*: To find out whether we can further increase the sample efficiency shown in Table 2, we train a random subset of games again on 10K, 25K, and 50K interactions with the full training budget that we used for the 100K interactions. In Figure 18 we see that this can lead to significant improvements in some cases, which could mean that the policy benefits from more training on imagined trajectories, but can even lead to worse performance in other cases, which could possibly be caused by overfitting of the world model. When the performance stays the same even with longer training, this could mean that better exploration in the real environment is required to get further improvements.

Table 2: Performance of our method at different stages of training compared with final scores of previous methods. We show individual game scores and mean human normalized scores. The normalized mean of our method is higher than SimPLe after only 25K interactions, and higher than previous methods after 50K interactions.

					TWM (ours)					
Game	Random	Human	SimPLe	SPR	5K	10K	25K	50K	75K	100K
Alien	227.8	7127.7	616.9	841.9	202.8	383.2	463.6	532.0	776.6	674.6
Amidar	5.8	1719.5	74.3	179.7	3.8	35.4	54.9	101.3	103.0	121.8
Assault	222.4	742.0	527.2	565.6	241.5	315.4	418.7	466.8	627.8	682.6
Asterix	210.0	8503.3	1128.3	962.5	277.0	297.0	536.0	912.0	886.0	1116.6
BankHeist	14.2	753.1	34.2	345.4	17.6	4.4	17.4	125.2	288.4	466.7
BattleZone	2360.0	37187.5	4031.2	14834.1	2640.0	3120.0	2700.0	3740.0	5260.0	5068.0
Boxing	0.1	12.1	7.8	35.7	0.8	3.4	28.5	60.1	67.1	77.5
Breakout	1.7	30.5	16.4	19.6	1.0	5.9	6.9	12.5	15.0	20.0
ChopperCommand	811.0	7387.8	979.4	946.3	928.0	1044.0	1358.0	1306.0	1438.0	1697.4
CrazyClimber	10780.5	35829.4	62583.6	36700.5	7425.0	14773.2	39456.8	45916.0	67766.2	71820.4
DemonAttack	152.1	1971.0	208.1	517.6	174.7	184.4	216.8	335.2	391.4	350.2
Freeway	0.0	29.6	16.7	19.3	0.0	4.6	20.8	23.7	23.9	24.3
Frostbite	65.2	4334.7	236.9	1170.7	66.2	204.6	297.8	247.6	1165.4	1475.6
Gopher	257.6	2412.5	596.8	660.6	345.2	414.0	593.2	1213.2	1549.2	1674.8
Hero	1027.0	30826.4	2656.6	5858.6	448.9	1552.6	4790.9	6302.7	9403.8	7254.0
Jamesbond	29.0	302.8	100.5	366.5	35.0	117.0	172.0	215.0	322.0	362.4
Kangaroo	52.0	3035.0	51.2	3617.4	28.0	92.0	476.0	724.0	876.0	1240.0
Krull	1598.0	2665.5	2204.8	3681.6	1763.6	2552.8	4234.0	4699.2	5848.0	6349.2
KungFuMaster	258.5	22736.3	14862.5	14783.2	574.0	16828.0	16368.0	17946.0	22936.0	24554.6
MsPacman	307.3	6951.6	1480.0	1318.4	245.9	535.1	1077.5	1224.3	1287.6	1588.4
Pong	-20.7	14.6	12.8	-5.4	-20.4	-19.8	-7.7	8.0	19.9	18.8
PrivateEye	24.9	69571.3	35.0	86.0	61.0	80.0	80.0	3.2	88.8	86.6
Qbert	163.9	13455.0	1288.8	866.3	151.0	298.5	703.5	1046.5	1788.5	3330.8
RoadRunner	11.5	7845.0	5640.6	12213.1	24.0	1120.0	5178.0	7436.0	8034.0	9109.0
Seaquest	68.4	42054.7	683.3	558.1	76.8	221.2	428.4	572.0	704.0	774.4
UpNDown	533.4	11693.2	3350.3	10859.2	385.8	1963.0	2905.6	4922.8	10478.6	15981.7
Normalized Mean	0.000	1.000	0.332	0.616	0.007	0.133	0.408	0.624	0.832	0.956

Wall-Clock Times: For each run, we give the agent a total training and evaluation budget of roughly 10 hours on a single NVIDIA A100 GPU. The time can vary slightly, since the budget is based on the number of updates. An NVIDIA GeForce RTX 3090 requires 12-13 hours for the same amount of training and evaluation. When using a vanilla transformer, which does not use the memory mechanism of the Transformer-XL architecture (Dai et al., 2019), the runtime is roughly 15.5 hours on an NVIDIA A100 GPU, i.e., 1.5 times higher.

We compare the runtime of our method with previous methods in Table 3. Our method is more than 20 times faster than SimPLe, but slower than model-free methods. However, our method should be as fast as other model-free methods during inference. In Table 2 we have shown that our method

Table 3: Approximate runtime (i.e., training and evaluation time for a single run) of our method compared with previous methods that also evaluate on the Atari 100k benchmark. Runtimes of previous methods are taken from Schwarzer et al. (2021). They used an improved version of DER (van Hasselt et al., 2019), which is roughly equivalent to DrQ (Yarats et al., 2021), so the specified runtime might differ from the original DER implementation. There are data augmented versions for SPR and DER. All runtimes are measured on a single NVIDIA P100 GPU.

Method	Model-based	Runtime in hours
SimPLe	✓	500
TWM (ours)	✓	23.3
SPR (with aug.)	✗	4.6
SPR (w/o aug.)	✗	3.0
DER/DrQ (with aug.)	✗	2.1
DER/DrQ (w/o aug.)	✗	1.4

achieves a higher human normalized score than previous sample-efficient methods after 50K interactions. This suggests that our method could potentially outperform previous methods with shorter training, which would take less than 23.3 hours.

To determine how time-consuming the individual parts of our method are, we investigate the throughput of the models, with the batch sizes of our main experiments. The Transformer-XL version is almost twice as fast, which again shows the importance of this design choice. The throughputs were measured on an NVIDIA A100 GPU and are given in (approximate) samples per second:

- World model training: 16,800 samples/s
- World model imagination (Transformer-XL): 39,000 samples/s
- World model imagination (vanilla): 19,900 samples/s
- Policy training: 700,000 samples/s

We also examine how fast the policy can run in an Atari game. We measured the (approximate) frames per second on a CPU (since the batch size is 1). Conditioning the policy on $[z, h]$ is about 3 times slower than z, since the transformer is required:

- Policy conditioned on z: 653 frames/s
- Policy conditioned on $[z, h]$: 213 frames/s

A.2 Derivation of Balanced Cross-Entropy Loss

Hafner et al. (2021) propose to use a balanced KL divergence loss to jointly optimize the observation encoder q_θ and state predictor p_θ with shared parameters θ, i.e.,

$$\lambda\, D_{\mathrm{KL}}(\mathrm{sg}(q_\theta) \parallel p_\theta) + (1-\lambda)\, D_{\mathrm{KL}}(q_\theta \parallel \mathrm{sg}(p_\theta)), \tag{7}$$

where $\mathrm{sg}(\cdot)$ denotes the stop-gradient operation and $\lambda \in [0, 1]$ controls how much the state predictor adapts to the observation encoder and vice versa. We use the identity $D_{\mathrm{KL}}(q \parallel p) = H(q, p) - H(q)$, where $H(q, p)$ is the cross-entropy of distribution p relative to distribution q, and show that our loss functions lead to the same gradients as the balanced KL objective, but with finer control over the individual components:

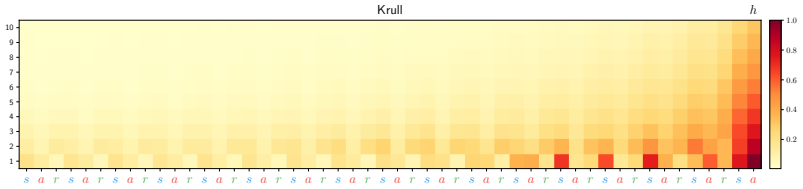
$$\nabla_\theta \left[\lambda\, D_{\mathrm{KL}}(\mathrm{sg}(q_\theta) \parallel p_\theta) + (1-\lambda)\, D_{\mathrm{KL}}(q_\theta \parallel \mathrm{sg}(p_\theta))\right] \tag{8}$$

$$= \nabla_\theta \left[\lambda \left(H(\mathrm{sg}(q_\theta), p_\theta) - H(\mathrm{sg}(q_\theta))\right) + (1-\lambda) \left(H(q_\theta, \mathrm{sg}(p_\theta)) - H(q_\theta)\right)\right] \tag{9}$$

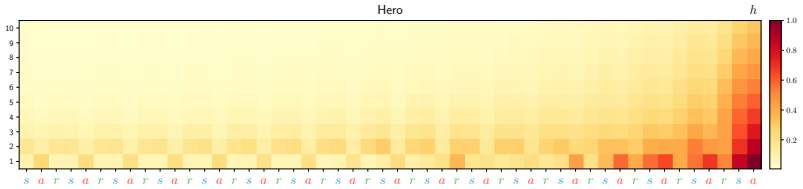
$$= \nabla_\theta \left[\lambda_1\, H(\mathrm{sg}(q_\theta), p_\theta) + \lambda_2\, H(q_\theta, \mathrm{sg}(p_\theta)) - \lambda_3\, H(q_\theta)\right], \tag{10}$$

since $\nabla_\theta H(\mathrm{sg}(q_\theta)) = 0$ and by defining $\lambda_1 = \lambda$ and $\lambda_2 = 1 - \lambda$ and $\lambda_3 = 1 - \lambda$. In this form, we have control over the cross-entropy of the state predictor relative to the observation encoder and vice versa. Moreover, we explicitly penalize the entropy of the observation encoder, instead of being entangled inside of the KL divergence.

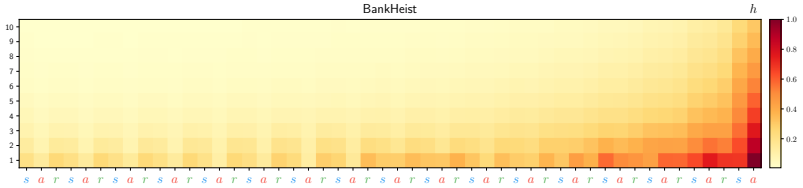
As common in the literature, we define the loss function by omitting the gradient in Equation (10), so that automatic differentiation computes this gradient. For our world model, we split the objective into two loss functions, as the observation encoder and state predictor have separate parameters, yielding Equations (3) and (4).



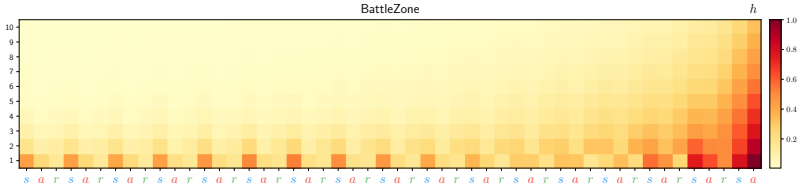
(a) This world model focuses on previous states.



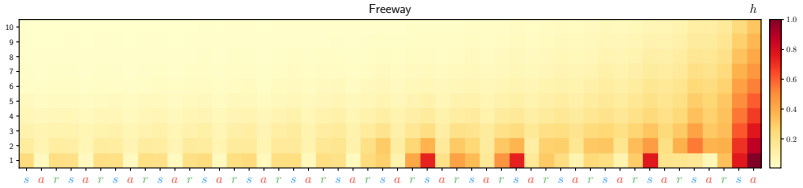
(b) This world model focuses on previous actions, indicating that the effect of actions can last longer than a single time step.



(c) This world model attends to all three modalities in the recent past.



(d) This world model attends to states at all time steps, probably because of the complexity of this 3D game.



(e) This world model mainly focuses on four states at specific time steps.

Figure 10: Average attention maps of the transformer, computed over many time steps. They show how different games require a different focus on modalities and time steps.

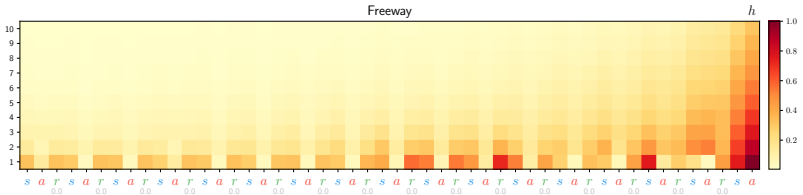
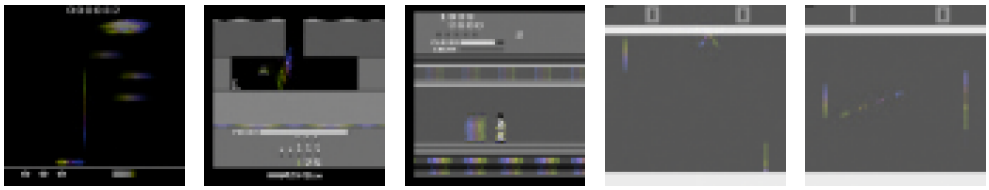


Figure 11: Attention map for Freeway for a single time step. At this point the player hits a car and gets pushed back (see also Figure 5b) and the world model puts more attention to past states and rewards, compared with the average attention at other time steps, as shown in Figure 10e. The world model has learned to handle this situation separately.

Figure 12: Three trajectories for the game KungFuMaster generated by our world model, using the same starting state. Because of its stochastic nature, the world model is able to generate three different situations (one opponent, two opponents, one other type of opponent). Note that we only show every third frame to cover more time steps.

Figure 13: A long trajectory imagined by our world model for the game Hero. The player traverses five different rooms and the world model is able to correctly predict the state and reward dynamics. Note that we only show every fifth frame to cover more time steps (the rewards lying in-between are summed up). The total number of time steps is 230.



(a) Assault (b) Hero (c) KungFuMaster (d) Pong (e) Pong

Figure 14: Visualization of frame stacks reconstructed from predicted states $\hat{z}_t$. Each frame in the stack is visualized by a different color. The world model is able to encode and predict movements.

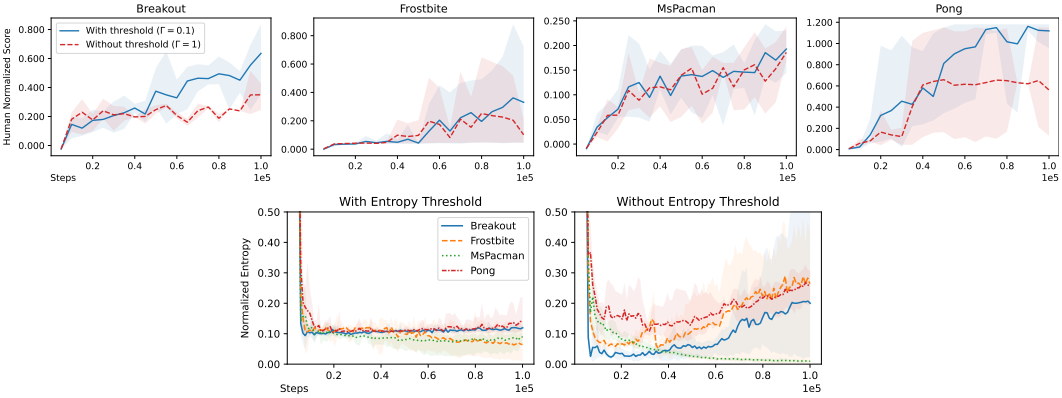


Figure 15: Effect of disabling the proposed thresholded entropy loss (by setting $\Gamma = 1$) on the performance and the entropy in a random subset of games. The thresholded version stabilizes the entropy and leads to a better score in Breakout and Pong, while the entropy behaves unfavorably without a threshold.

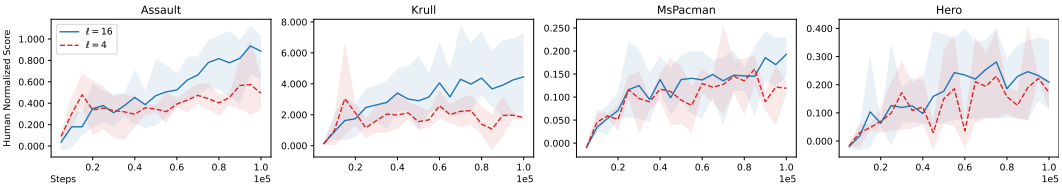


Figure 16: Comparison of the history length $\ell = 16$ used in our main experiments with $\ell = 4$ on a random subset of games. We observe a lower human normalized score for $\ell = 4$.

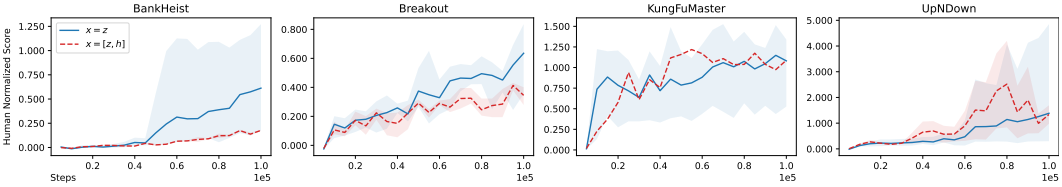


Figure 17: Conditioning the policy on $[z, h]$ compared with the usual z. In some cases the performance can be better during training, but the final score is lower or equal.

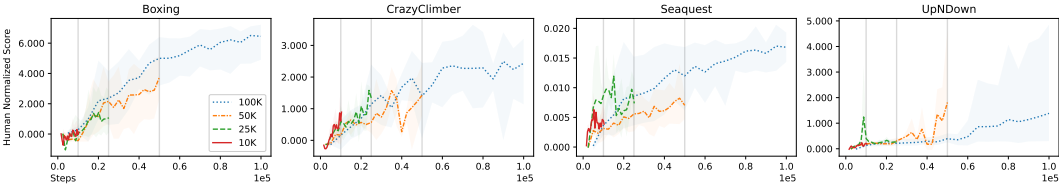


Figure 18: Scores on a random subset of games when we train with a lower number of interactions but the same training budget. This only leads to a significant improvement for UpNDown, where the final score is higher with only 50K interactions.

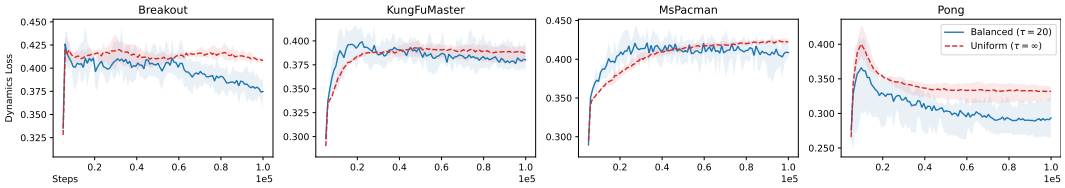


Figure 19: Comparison of the proposed balanced sampling procedure with uniform sampling. It shows the development of the dynamics loss from Equation (4), which is lower at the end of training in all cases.

A.3 Additional Training Details

In Algorithm 1 we present pseudocode for training the world model and the actor-critic agent. We use the SiLU activation function (Elfwing et al., 2018) for all models. In Table 4 we summarize all hyperparameters that we used in our experiments. In Table 5 we provide the number of parameters of our models.

Pretraining for Better Initialization: During training we need to correctly balance the amount of world model training and policy training, since the policy has to keep up with the distributional shift of the latent space. However, we can spend some extra training time on the world model with pre-collected data (included in the 100K interactions) at the beginning of training in order to obtain a reasonable initialization for the latent states.

Algorithm 1 Training the world model and the actor-critic agent.

```
function train_world_model()
  // sample sequences of observations,
  // rewards, actions and discounts
  o,a,r,d = sample_from_dataset()
  z = encode(o)
  o_hat = decode(z)
  h = transformer(z,a,r)
  r_hat,d_hat,z_hat = predict(h)

  // optimize world model via
  // self-supervised learning
  optim_observation(o,z,o_hat,z_hat)
  optim_dynamics(r,d,z,r_hat,d_hat,z_hat)

  // z will be used for imagination
  return z
```

```
function train_actor_critic(z)
  // imagine trajectories of states,
  // rewards, actions and discounts;
  // use z as starting point
  imag = [z]
  for t = 0 until H do
    a = actor(z)
    imag.append(a)
    h = transformer(imag)
    r,d,z = predict(h)
    imag.extend([r,d,z])

  // optimize actor-critic via
  // reinforcement learning
  optim_actor_critic(imag)
```

Table 4: Hyperparameters used in our experiments.

Description	Symbol	Value
Dataset sampling temperature	τ	20
Discount factor	γ	0.99
GAE parameter	λ	0.95
World model batch size	N	100
History length	ℓ	16
Imagination batch size	M	400
Imagination horizon	H	15
Encoder entropy coefficient	α_1	5.0
Consistency loss coefficient	α_2	0.01
Reward coefficient	β_1	10.0
Discount coefficient	β_2	50.0
Actor entropy coefficient	η	0.01
Actor entropy threshold	Γ	0.1
Environment steps	—	100K
Frame skip	—	4
Frame down-sampling	—	64×64
Frame gray-scaling	—	Yes
Frame stack	—	4
Terminate on live loss	—	Yes
Max frames per episode	—	108K
Max no-ops	—	30
Observation learning rate	—	0.0001
Dynamics learning rate	—	0.0001
Actor learning rate	—	0.0001
Critic learning rate	—	0.00001
Transformer embedding size	—	256
Transformer layers	—	10
Transformer heads	—	4×64
Transformer feedforward size	—	1024
Latent state predictor units	—	4×512
Reward predictor units	—	4×256
Discount predictor units	—	4×256
Actor units	—	4×512
Critic units	—	4×512
Activation function	—	SiLU

Table 5: Number of parameters of our models.

Model	Symbol	# Parameters
Observation model	ϕ	8.2M
Dynamics model	ψ	10.8M
Actor	θ	1.3M
Critic	ξ	1.3M
World model	—	19M
Actor-critic	—	2.6M
Total	—	21.6M
Encoder + actor (at inference time)	—	4.4M