BOXLCD: A SIMPLE TESTBED FOR LEARNED SIMULATOR RESEARCH

Matthew Wilson
Department of Computer Science
University of British Columbia
Vancouver, Canada
mattwilsonmbw@gmail.com

ABSTRACT

Traditional robotics simulators are inflexible and limited in what they can represent and simulate. Data-driven, or *learned simulators*, seem a promising way forward. However, modeling real world physical systems is challenging and progress in this space remains limited. In this work we introduce boxLCD, a simplified testbed for developing ideas and algorithms around learned robotics simulators and their usage. We provide results on a set of sample environments, both for learning predictive physics models and for using those models as simulators to train reinforcement learning agents to achieve simple goals. The open source project with example code, training scripts, and videos, can be found at: github.com/matwilso/boxLCD.

1 INTRODUCTION

Simulators are used universally in robotics to develop and debug code which is then run in the physical world. This enables quicker iteration cycles and helps mitigate robot and environment damage, among other things. More recently, simulators have also been used as a source of data for training deep learning vision models and policies that then operate in the real world (Sadeghi & Levine, 2017; Tobin et al., 2017; James et al., 2017; Wilson & Hermans, 2019; Chebotar et al., 2019; OpenAI et al., 2020). Sim2real learning provides several advantages over real world training in speed, safety, and environment read + write access. However, the final performance of sim2real relies on both the accuracy of the simulator and the ability to transfer knowledge across whatever reality gap remains. Absent major progress in transfer learning, improving the accuracy of simulators is of crucial importance to the future of sim2real—and perhaps robotics more generally. After all, the closer that simulators approximate the real world, the more effectively developers can validate systems and ensure desired behavior in real world deployment.

Despite the promise of simulation, traditional robotics simulators remain inflexible and limited in what they can accurately represent and simulate. Users must arduously specify the many details of the environment they would like to simulate—a haphazard process that often requires substantial manual calibration effort (Tan et al., 2018; OpenAI et al., 2020) and still leads to a crude approximation. Furthermore, current simulators are often incapable of simulating the relevant physics, no

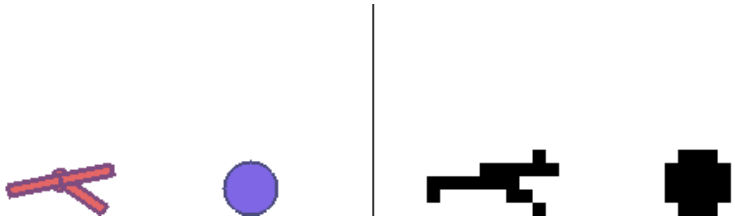


Figure 1: boxLCD sample environment. Left: standard RGB frame of a scene. Right: low-resolution and binarized frame of the same scene, reminscent of a primitive Liquid Crystal Display (LCD) view.

matter how the parameters are set. Current simulators do not represent processes like melting and shattering, or strange materials like fire, paint, light switches, and microwaves, for example, and it would require painstaking domain-specific effort to add support for these in the current paradigm.

Data-driven, or *learned simulation*, seem a promising way forward in developing more accurate and usable systems. Traditional simulators already rely on data—just indirectly. Humans observe the world, develop understandings, distill these into equations, and program the equations into computer software. The bottleneck arises in the human understanding and ingenuity required to drive this feedback loop. Learned simulation can help shortcut this by directly consuming data, optimizing models to fit the data with gradient descent, and directly producing accurate, flexible, and differentiable physics software for debugging, evaluating, and training robotic policies.

To date, there has been some work on learning parts of simulators from real world data and using them for training. Hwangbo et al. (2019) and Lee et al. (2020) learned a model of a series elastic actuator on the Anymal Robot and used this in the simulator loop for sim2real. Others have worked on hybrid learned differentiable simulators (Heiden et al., 2020), or cloning traditional simulator using graph neural networks to enable planning (Sanchez-Gonzalez et al., 2018; Li et al., 2019). The scope of these systems, however, remains limited.

Developing a fully learned simulator from real world data represents a massive undertaking and research effort. Our philosophy in this work is that it is best to start small, build up strong intuitions, and gradually increase the difficulty of the challenges as progress is made—while always remaining focused on the final goal of systems that work in the real world. This motivates the idea of boxLCD, which aims to provide a close analogue to the real world problems of learning a robotics simulator, while remaining extremely tractable to work on. In this paper, we describe the design decisions behind boxLCD, and we provide sample environments and experimental results for learning models of these environments and using those models as learned simulators for reinforcement learning.

2 BOXLCD DESIGN CHOICES

2.1 EMULATING REAL WORLD CHALLENGES

boxLCD aims to serve as a testbed that accurately captures the challenge of building learned robotics simulators. It is thus built to satisfy several requirements:

Physics-based and actuator-based. The real world has consistent physics, with forces like gravity and friction. Robots do not move magically; they must coordinate their many actuators to move their bodies in specific ways. Movements like "forward" and "backward" are not primitives; they are built from joint movements. This is reflected in the testbed through the use of the box2D physics engine and the design of the robots and environments.

Pixel-based. Humans and robots primarily sense the real world through vision (pixels). boxLCD is largely focused around pixel-based sensing and predicting physics by predicting future video.

Multi-modal. Robots have sensors like joint encoders and inertial measurement units (IMUs) that provide complementary information to vision. boxLCD provides support for proprioceptive information and other modalities.

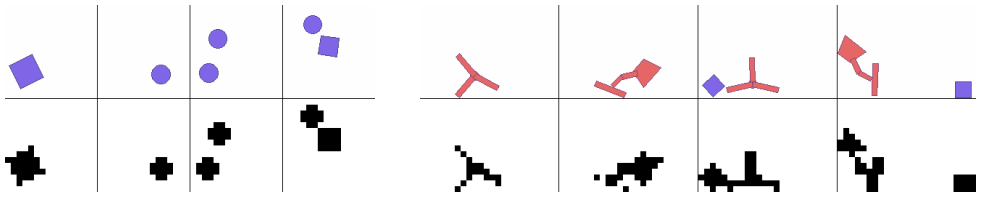
Partially observable. Sensors do not tell the full story of the world. Robots constantly have to make estimates of the underlying state that they are observing evidence of. boxLCD supports this by providing only proprioception and low-res images, not direct observations of the environment.

Interfaceable. A desirable property of future learned simulators is that they be easy to interface with, for example by letting a user specify scenes by providing structured and unstructured information like meshes, natural language, and video frames. The baseline models we train (Section 3.2) demonstrate the ability to take in and complete a video prompt, but it is interesting future work to experiment with other modalities and description formats as well.

2.2 REMAINING TRACTABLE

At the same time, boxLCD aims to remain computationally tractable and easy to work with:

2D physics. box2D deals with things like contacts, friction, gravity, but it is much simpler and lower-dimensional than the real world or 3D simulators like Mujoco and Bullet.



(a) Tier 0 Envs: Passive Objects (b) Tier 1 Envs: Simple Robot and Manipulation

Figure 2: boxLCD environments we evaluate on, in both full resolution color and basic LCD rendering. Left to right: Dropbox, Bounce, Bounce2, Object2, Urchin, Luxo, UrchinCube, LuxoCube.

Low-resolution rendering. boxLCD enables support for both color and high-dimensional image frames, but the sample environments use at most a 16x32 = 512 sized binary images (smaller than MNIST 28x28 = 784). This greatly reduces the computational cost to process the image data, for example eliminating the need to model complex textures and other high-frequency image details. (It does however lead to ambiguities in object pose that can make prediction and control more difficult.)

Procedurally generated and customizable. boxLCD enables users to create custom new scenarios and generate as much data as they need. The programmability of these environments also makes it easy to evaluate RL policies in different settings (as shown in Section 3.3).

2.3 Related work

While other benchmarks provide some similar features, we are not aware of any work with the same goals as boxLCD, nor ones that simultaneously satisfies the same criteria. For example, PHYRE (Bakhtin et al., 2019) and Simulated Billiards (Qi et al., 2021) use simple 2D physics systems, but focus on the much simpler case of taking only a single action at the start of the episode. Other environments are either 3D and much more complex to model and/or do not simulate physics or robots (Young & Tian, 2019; Xia et al., 2018; Savva et al., 2019; Beattie et al., 2016).

3 Experiments

To demonstrate what is possible with boxLCD, we present results on learning models (Section 3.2) and using those models as learned simulators for RL (Section 3.3). We describe the environments, our models, RL approaches, and results below. Code to reproduce this can be found at: github.com/matwilso/boxLCD.

3.1 Sample environments

We provide a few sample environments, ranging from fairly trivial to moderately challenging. These environments deal progressively with: passive object and environment interactions, robot actuation and action-conditioning, and finally robot+object interactions. We provide still frame images in Figure 2, and descriptions for each env below with indications of (EP_LEN x HEIGHT x WIDTH):

- **Dropbox (25x16x16):** A large box with random initial XY coordinates and angle.
- **Bounce (50x16x16):** A ball with higher restitution (bounciness) than the box, which tends to bounce a few times before coming to rest.
- **Bounce2 (50x16x16):** Two balls from Bounce; induces multi-object interactions.
- **Object2 (50x16x16):** Two random objects (ball or box); requires remembering identities.
- **Urchin (100x16x32):** A symmetric robot with 3 actuated limbs. More challening; requires conditioning on actions at every time-step, and enables body goals.
- **Luxo (100x16x32):** Like Urchin, but a lamp-shaped robot with 3 actuated limbs.
- **UrchinCube (150x16x24):** Urchin in an environment with a cube; induces interactions between a joint-actuated robot and a passive object and enables manipulation goals.
- **LuxoCube (150x16x24):** Like UrchinCube but with Luxo.

	Dropbox		**Bounce**		**Bounce2**		**Object2**		**Urchin**		**Luxo**		**UrchinCube**		**LuxoCube**	
	RSSM	FIT	RSSM	FIT	RSSM	FIT	RSSM	FIT	RSSM	FBT	RSSM	FBT	RSSM	FBT	RSSM	FBT
FVD*	**35.1**	**1.6**	**72.2**	**0.1**	**253.8**	**5.7**	**130.4**	**20.4**	**263.4**	**2.3**	**307.2**	**78.2**	**311.9**	**5.7**	**394.7**	**101.6**
SSIM	0.795	0.920	0.808	0.977	0.652	0.837	0.574	0.669	0.716	0.726	0.653	0.657	0.614	0.626	0.531	0.531
PSNR	31.04	51.67	47.58	86.33	20.41	41.31	12.39	14.70	13.30	13.52	11.79	11.85	11.21	11.42	9.802	9.761
COS*	0.210	0.082	0.270	0.023	0.420	0.190	0.555	0.443	0.659	0.297	0.677	0.412	0.644	0.360	0.653	0.467

Table 1: Quantitative comparison of prompted model predictions on envs. FVD* metrics are bolded because we believe it represents the best metric for quality of samples. The Transformer models perform the best across all metrics and environments, but we make no strong claims that a tuned RSSM, with discrete latents as in Hafner et al. (2020b), could not perform as well. FVD* (based on (Unterthiner et al., 2019)) and COS* (based on (Zhang et al., 2018)) are metrics based on learned autoencoder features trained for each env. COS stands for cosine distance between base simulator and learned simulator predictions in a feature space.

3.2 Model Learning

We compare performance on learning to predict future frames using two models: an RSSM from Dreamer v1 Hafner et al. (2020a) and a custom autoregressive Transformer approach. We choose RSSMs as they are a recent state of the art model for dealing with images and stochasticity in predictions. And we choose to use Transformers because they have been shown to be powerful and scalable in sequence prediction and multi-modal generative modeling tasks (Brown et al., 2020; Ramesh et al., 2021; Henighan et al., 2020), of which physics simulation is an example.

Details. We transcribed the RSSM from TensorFlow to PyTorch and modified it to take in and predict proprioceptive information. The autoregressive Transformer is a custom design, with two-variants: Flat Image Transformer (FIT) and Flat Binary Transformer (FBT). FIT just flattens the image at each time step and uses that as the token for the Transformer. To train our model, we feed those flat image tokens in; the model produces independent Bernoulli distributions for each pixel in the frame; and we optimize these distributions to match the ground truth (Loss $= -\log p(\mathbf{x})$). To sample the model, we prompt it with the beginning 1-3 frames of the episode, and have it predict the rest autoregressively. FBT is similar, but pre-trains an autoencoder to embed both image and proprioception into a latent binary space, and it uses this latent binary space as tokens instead. FIT is a naive approach to demonstrate the tractability of boxLCD, while FBT is more likely to scale to higher dimensional color image spaces. We make no claims about the advantage of this type of approach, but we find they are simple, train relatively quickly, and tend to work well.

Results. We train models on datasets on 100k rollouts, with a test set of 10k rollouts. We evaluate RSSM on all environment, and evaluate FIT on the environments without proprioceptive data. Results are shown in Table 1. Example predictions are show in Figure 3 and the Appendix. Video samples and more details on how COS* and FVD* are computed can be found in the repo.

These results demonstrate the tractability of boxLCD, but also the challenges it provides. Simple boxLCD tasks like Bounce and Dropbox can be solved with very little compute using a naive approach. And while some progress can be made on the multi-object and robot-object tasks, the losses

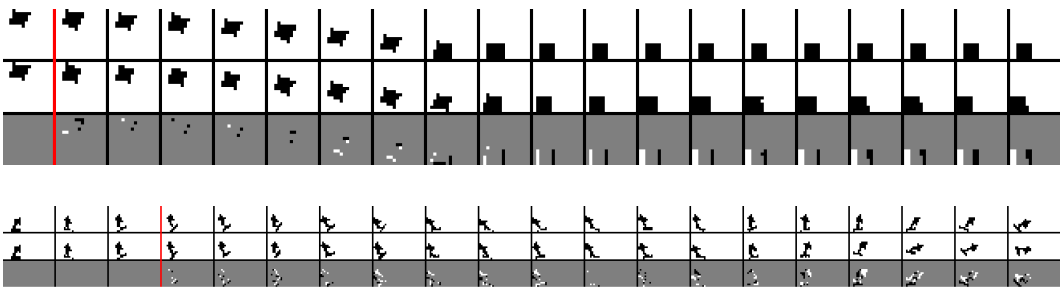
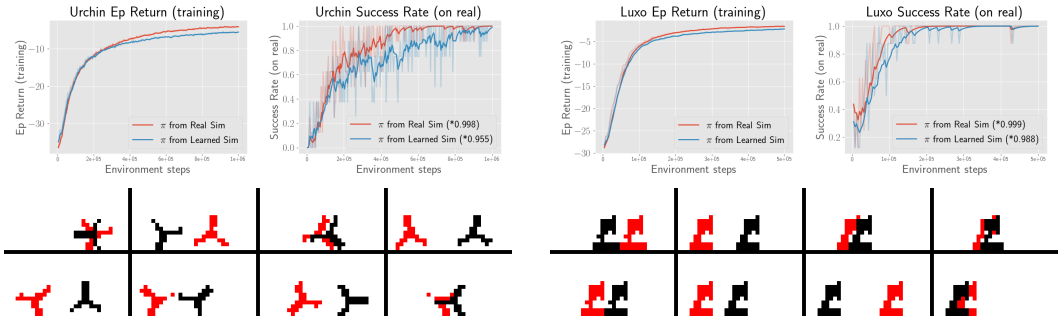


Figure 3: Results for training the baseline approach with different number of parameters.



(a) Urchin BodyGoalEnv (b) Luxo BodyGoalEnv

Figure 4: Results for BodyGoal tasks. Initial states are in black, goals are in red. Rewards are proportional to the current distance to the goal and episodes terminate after T steps or when the agent reaches the goal within some tolerance ϵ. We train agents in both the "real simulator" and the learned simulator. Episode return is reported for the environment used to train in (i.e., for learned simulator, rewards are computed based on predicted state). Success rate is reported on the real environment. The curves are noisy, but the * values represent evaluating at convergence with N=1000. We find that the learned simulator policy transfers nearly perfectly to the real simulator.

and sample quality remain poorer. The model has a bit of trouble with object permanence and retaining the identity of objects when they are subject to change, showing plenty of room for improvement with more sophisticated methods, for example using stronger spatial and relational inductive biases ((Battaglia et al., 2018)) to deal with the symmetry in the Urchin robot or the multi-object settings.

3.3 Reinforcement Learning Inside the Learned Simulator

Finally, the best indicator of model usefulness is whether it can be used to learn to solve tasks. By default, boxLCD does not provide rewards, but it is designed around the standard Gym Environment interface (Brockman et al., 2016) and so supports wrapping environments with a reward definition. We have implemented "BodyGoals" and "CubeGoals", where the objective is to reach a proprioceptive or object location goal state. Reward is given proportional to distance to the goal and episodes are terminated after T steps, or when reaching the goal within a certain threshold.

We demonstrate how our predictive models of Urchin and Luxo can be used as learned simulators to solve BodyGoals using PPO (Schulman et al., 2017). We train agents in both the learned simulator and the "real simulator, and find that agents trained in the learned simulator can perform nearly as well when evaluated in the real simulator. See results in Figure 4. For these tests, all agents operate on proprio state information, not images. Agents train in 40 minutes to about 3 hours, with learned simulation taking about 50% longer (without any optimization effort, e.g., by keeping data on GPU). We note that the CubeGoal environments are more challenging, as they require complex body coordination for these robots to move the object, and we have only solved UrchinCube so far using full state information about the object location.

4 Conclusion and Future Work

boxLCD aims to serve as a testbed for learning robotics simulators. Ultimately, the goal is to learn models in the real world that end up helping us solve real world problems. We believe it is still early days for video prediction, generative modeling, and learned simulation, and that this testbed will help provide greater traction on these problems.

boxLCD is still in active development. The current sample environments are useful for developing learned simulator models and evaluating them on simple RL tasks. But in the future, we plan to add more complex environments and agents, additional modalities, and other features to emulate what will be necessary in developing real learned simulators.

REFERENCES

Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. Phyre: A new benchmark for physical reasoning. 2019.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979. IEEE, 2019.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2020a.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models, 2020b.

Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. Neuralsim: Augmenting differentiable simulators with neural networks. *arXiv preprint arXiv:2011.04217*, 2020.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling, 2020.

Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

Stephen James, Andrew J Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *Conference on Robot Learning*, pp. 334–343. PMLR, 2017.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Haozhi Qi, Xiaolong Wang, Deepak Pathak, Yi Ma, and Jitendra Malik. Learning long-term visual dynamics with region proposal interaction networks. In *ICLR*, 2021.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *Robotic: Science and Systems (RSS)*, 2017.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4470–4479. PMLR, 10–15 Jul 2018. URL http://proceedings.mlr.press/v80/sanchez-gonzalez18a.html.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9339–9347, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

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)*, pp. 23–30. IEEE, 2017.

Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges, 2019.

Matthew Wilson and Tucker Hermans. Learning to manipulate object collections using grounded state representations. *Conference on Robot Learning*, 2019.

Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson Env: real-world perception for embodied agents. In *Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.

A Appendix

A.1 More model samples

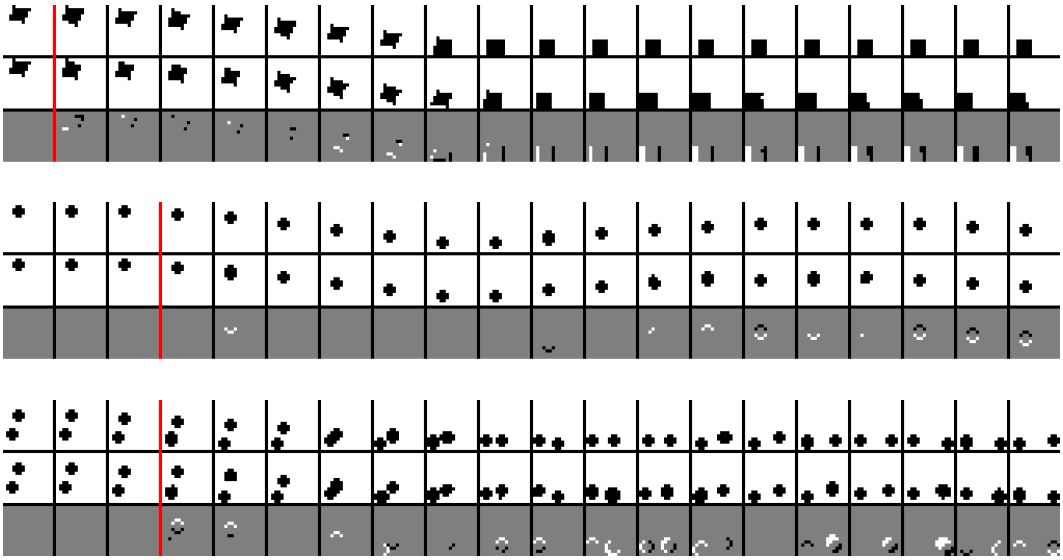


Figure 5: Extra Tier 0 environment model predictions from the RSSM model.

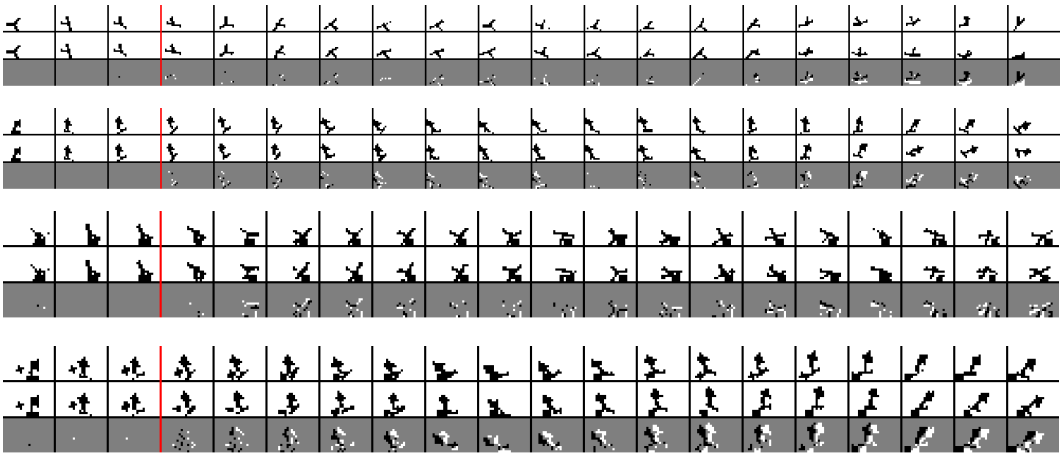


Figure 6: Extra Tier 1 environment model predictions from the FBT model.