Learning World Models for puzzle like games

Théo Michel, Svetlana Levit

Project Report : Learning World Models for puzzle like games

Théo MICHEL
tmichel003@enseirb-matmeca.fr
Supervisor : Svetlana Levit

1 Abstract

This work was part of the course Learning and Intelligent Systems during the winter semester 2022-2023 at the TU Berlin.

Recently, we have seen a rise of model-based algorithms in reinforcement learning, whose main advantage lies in the training efficiency enabled by representing the world in a compressed latent representation. One such algorithm is Dreamer V2, which has surpassed human-level performance on all Atari games, and has also been used to teach a quadruped to walk using only real-world experience [7]. However, we have not seen the usefulness of the Dreamer V2 algorithm when applied to more complicated logical scenarios, and we cannot really predict the results because we do not really have a deep understanding of the world model. The following is a summary article of the exploration of the Dreamer V2 algorithm applied to complex tasks. We will examine the results of the algorithm when applied to different tasks (puzzles, exploration game...) with different levels of difficulty and logical complexity. And we will also examine the quality of the generated world models, using simple methods to analyse input and output correlations, and laying the groundwork for more advanced world model analysis.

2 Introduction

Dreamer V2 is the successor to the Dreamer algorithm, which first introduced the idea of learning the policy only in the latent space representation of the world as such, allowing for more sample-efficient and faster training [6]. The world model is learned using only the pixels of the image, but can also be used for categorical values. The general aim was to look at sample-efficient algorithms for world model learning, this was done in the context of research on solving continuous 3D puzzles where agents are sometimes required to build tools from multiple parts. Sample efficient world models can help to solve these problems, to evaluate the model produced by the Dreamer V2 algorithm the following objectives were set:

- Reproduce the training of the Dreamer V2 paper
- Extend the experimentation to puzzle like games
- Interpret the latent variables of the world model
- Apply the method to the 3D continuous control simulation RAI were the agent has to solve 3d puzzles involving tool building,
- Analyze related papers (DayDreamer, DreamerV3, ...).

3 Background and related work

Reinforcement learning is the study of how to solve problems using rewards for taking certain actions. World model learning has long been an active area of study in reinforcement learning, with

5th Conference on Robot Learning (CoRL 2021), London, UK.

many theorising that model-based algorithms should be the best for reinforcement learning, but for a long time model-free algorithms have dominated the benchmarks.

3.1 Theoretical Background

3.1.1 Soft-Actor critic

This algorithm allows the agent to learn the optimal policy, introduced in the following paper [9]. An actor-critic policy learning algorithm consists of an actor that chooses an action a based on the state of the environment s, and a critic that evaluates the state-action pair (a, s). They then work and learn together to get the best performance possible. The problem is that they don't explore enough, so they get stuck in local optima and don't generalise well, which is where the "soft" part comes in. "Soft" means that the policy is trained to maximise a trade-off between expected return and entropy, so that it encourages exploration and at the same time helps generalisation, as the agent will find itself in unknown situations more often.

The soft-actor critic is in our case helped by the "world model".

3.1.2 Model Based and Model Free RL

Model based reinforcement learning is a paradigm in RL in which a world model is typically learned by observing the state transitions and rewards that occur as the agent interacts with the environment. The agent uses this data to learn a model that predicts the next state h_{i+1} and reward r_{i+1}given the current state and action.

This model can be used by the agent to simulate future states of the environment and plan its actions accordingly, and also as in our case to train in it more efficiently. The world model prediction often takes the shape of a Neural Network taking as input the state of the world h_i and action a and predicting the next state h_{i+1} and the next reward r_{i+1}.

Model Free RL on the other hand doesn't use any form of world model, as a result it is often less sample efficient.

3.1.3 RSSM in reinforcement learning

Another important piece of the puzzle is the Recurrent State-space Model (RSSM) included in the algorithm. In an RSSM, the state of the system is represented as a set of latent variables that are updated over time based on a set of observations. This is useful for the agent to remember the actions it has taken and the information it has gathered in the past.

3.1.4 States and latent spaces

In the following paragraphs we will discuss two different states, the state of the environment, which is for example the pixel values displayed by the screen when playing a game, the values of the variables stored in ram, etc ... Also discussed are the hidden states h and z. These are commonly said to contain variables in latent space, i.e. an abstract compressed space where similar objects are close together. In our example, z contains discrete variables and h continuous variables. These states are used to compress data, and by forcing the compression of data into a more compact representation, we often aim to extract only the useful features.

3.2 Related works

In the last five years, a lot of papers showing the possibility of efficient world model learning have been coming out see "Connected Papers" [12], in particular through relatively large RNNs as was introduced in the following paper [11]. And these models also have the advantage of being interpretable by asking them to output their future predictions, see the interpretation: [13].

At the time of choosing the algorithm the Dreamer V2 introduced in the paper "Mastering Atari with Discrete World Models" [1] was the best performing model based algorithm on a variety of tasks,

and was shown to be transferable to real world tasks, for example a four legged robot in the "Day Dreamer"[7] paper. It is also interesting to note that the third iteration of the Dreamer algorithm [14] also was released in January 2023, this version improved performance even without parameter tuning and solved the task of mining diamonds in Minecraft.

4 Methods

4.1 Presentation of the Dreamer V2 Algorithm

Recurrent model:	$h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1})$
Representation model:	$z_t \sim (z_t \mid h_t, x_t)$
Transition predictor:	$\hat{z}_t \sim (\hat{z}_t \mid h_t)$
Image predictor:	$\hat{x}_t \sim (\hat{x}_t \mid h_t, z_t)$
Reward predictor:	$\hat{r}_t \sim (\hat{r}_t \mid h_t, z_t)$
Discount predictor:	$\hat{\gamma}_t \sim (\hat{\gamma}_t \mid h_t, z_t)$.

All components are implemented as neural networks and ϕ describes their combined parameter vector. The transition predictor guesses the next model state only from the current model state and the action but without using the next image, so that we can later learn behaviors by predicting sequences of model states without having to observe or generate images. The discount predictor lets us estimate the probability of an episode ending when learning behaviors from model predictions. (Hafner et al., 2020)

The Dreamer V2 algorithm is a model-based reinforcement learning approach consisting of three main steps:

1. Learning a world model from experience (World Model learning)
2. Learning an actor and critic from imagined sequences of compact model states (Policy learning)
3. Executing the actor in the environment to grow the experience dataset (not detailed here)

The Dreamer V2 algorithm builds upon the world model that was introduced by PlaNet[8] and used in Dreamer V1 [6], by replacing its Gaussian latent variables with categorical variables.

4.1.1 World Model learning

The world model is trained from the agent's growing dataset of past experience that contains sequences of images, actions, rewards, and discount factors. The world model consists of an image encoder, a Recurrent State-Space Model (RSSM) to learn the dynamics, and predictors for the image, reward, and discount factor.

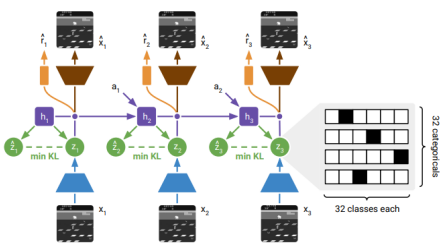


Figure 1: Visual representation of the world model learning (Hafner et al., 2020)[1]

4.1.2 Policy learning

The learned prior is used for imagination, and the KL-Loss both trains the prior and regularizes how much information the posterior incorporates from the image. The transition predictor guesses the next model state only from the current model state and the action but without using the next image, and the discount predictor lets us estimate the probability of an episode ending when learning behaviors from model predictions.

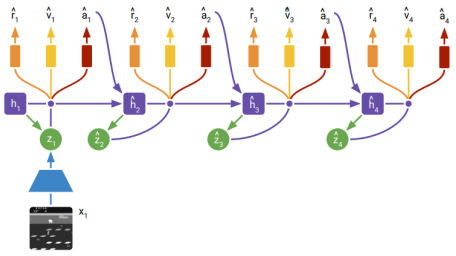


Figure 2: Visual representation of the dreaming (Hafner et al., 2020)[1][1]

4.1.3 World Model interception

Next we have to define what the world model is, and where we can intercept it and analyze/modify it. The world model is mainly represented by the RSSM giving $h_t, z_t, \hat{z}_t$ that are responsible for the memorization of the past, and the understanding of the logic. The other components of the world model $\hat{x}_t, \hat{r}_t, \hat{\gamma}_t$ are helpers to the RSSM that encode the input image, or decode the output from and to the h_t latent space, enabling the RSSM to learn more efficiently.

Considering the world model architecture seen in Fig.1 it is clear that h_t and z_t are the most interesting states to analyze. h_t being the memory of all past events and the state of the game, and z_t being the encoding of the observed world used to create h_t. It is important to note that z_t is a 32×32 discrete latent space, that is extremely compact whereas h_t is big : Horizon depth $\times$ Action space $\times$ $\|$Past Actions$\|$ $\times$ $\|$IWAE samples$\|$ so it was easier to focus on the z_t component at first.

In the following paragraphs will be discussed the influence of the z_t component on the h_t state, and see the impact of the modification of the z_t on the next predictions.

4.2 Implementation of the algorithm

A first attempt was made using the original repository [4], the code is written in Tensorflow, but it was hard to make it work as a lot of libraries were already deprecated. So a good alternative was to use the implementation of Dreamer V2 by Jurgis Pasukonis [3] (one of the authors of the Dreamer V3 paper), the implementation was made using Pytorch and had really similar results to the original paper, it was used as it was more recent, furthermore Pytorch felt more convenient for this particular task. The algorithm is identical to the original except for a few tweaks the main one being the use for training of GAE instead of using $TD - \lambda$, other than that the Pytorch implementation adds interesting features like :

- Multi-sample variational bound
- Categorical reward decoder
- Probe head for global map prediction
- Multiple workers

It does lack the capability of using Plan2Explore to explore the world model.

4.3 Extension to puzzle games

To extend to puzzle games, custom gym environments had to be written, see code [5]. Those can be found in

`/pydreamer/envs`

During the experimentation multiple games were tested, this is a list of the games that worked :

- `python launch.py --config defaults minigrid --env_id MiniGrid-DoorKey-8x8-v0 --probe_model`

Minigrid working :

- Emptyenv
- Doorkeyenv
- Fourroomsenv

Some minigrid environments require uncommenting parts of the code, see Readme for more detail, as the minigrid is gym designed for square input.

Not Working

- Crossing
- Dynamicobstaclesenv
- MiniGrid-MultiRoom-N2-S4-v0
- MiniGrid-Unlock-v0

Visit link for more detail. Other games that have been tested

- `python launch.py --configs defaults atari --env_id Atari-Alien-V5`
- `python launch.py --configs defaults atari --env_id Atari-Breakout`
- `python launch.py --configs defaults atari --env_id Atari-Pong`
- `python extract_model.py --configs defaults atari --env_id Atari-Adventure-v5`

The games will be explained more in detail in the next section.

5 Latent space interpretation methodology

There are multiple latent spaces that can be looked into for interpretation. In our case we have measured the influence of modifying z on h and the output image decoded from h.

The z state was capture by adding a pickle save of the state during training, then we could modify this z, and feed it back to the world model, where it will get combined with the usual h (also captured previously) and will use the decoder to create the image prediction of the world. The resulting image is used to try to infer logic in the world model. Then the image output given by the original state and the image output given by the modified are compared $\hat{x} - \hat{x}'$. This is an extremely simple approach, and it would be interesting to explore more advanced methods. See the results section 6.5

6 Experimental Results

6.1 Pong

Pong is a simple game where two pallets are sending the ball, back and forth, with each pallet controlled by a player, see Fig.5. Here the dream has to predict two logical consequences. The score increasing, and the bounce and trajectory of the ball. It is also interesting to look at the speed of learning, that is extremely fast see Fig.4, in just a few thousand steps the agent already has a convincing model of the world see Fig.11 with a loss rapidly decreasing see Fig.3.

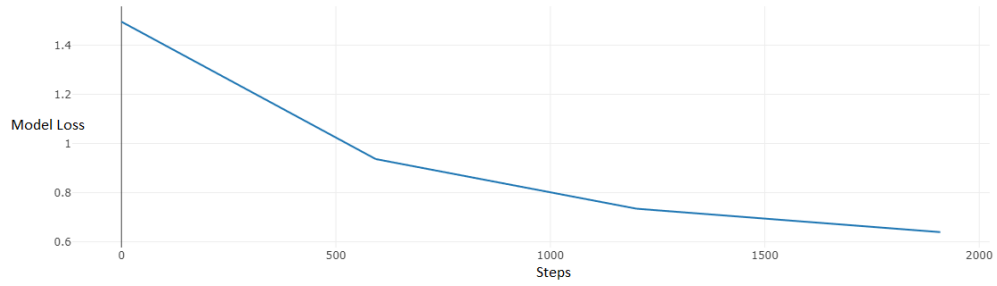


Figure 3: World Model loss value of the agent while training on the ”Pong” environment

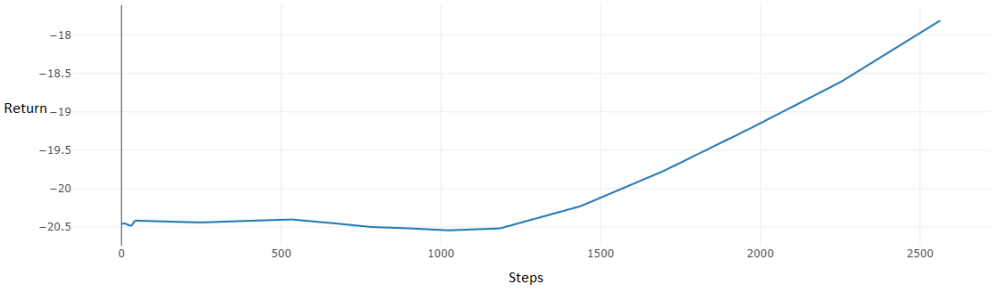


Figure 4: Average Cumulative-return value of the agent while training on the ”Pong” environment

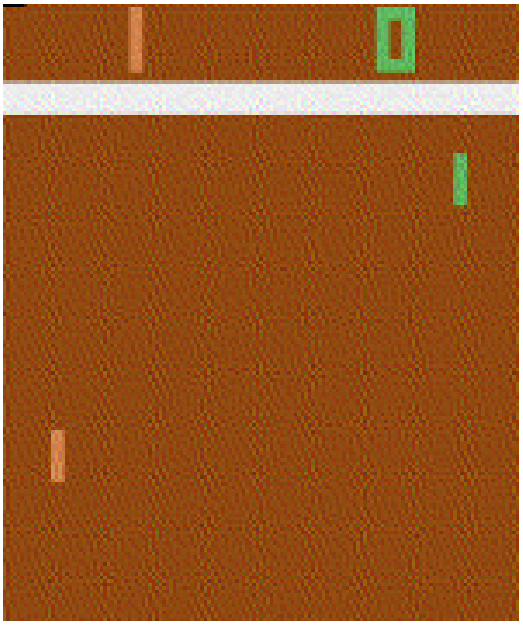


Figure 5: Example of the pong game

6.2 Minigrid

Minigrid is a grid based puzzle game with several different levels see Fig.7, it is interesting because it has sparse rewards, it has a partial observation space and the observation space is categorical instead of pixel based, but the algorithm is still successful. Here the interactions between objects are interesting, for example how a key opens a door. The agent learned perfectly in a few days. The training checkpoint was lost, and due to time and material constraints, the results could only be replicated in 80,000 steps, which takes an hour on a GPU. However, the same pattern as for the larger training can still be seen in Fig.6, i.e. multiple bursts of learning, getting larger each time, separated by periods of lower returns. This is due to the limited number of steps to solve the puzzle, the fact that the puzzle varies randomly in layout, and the fact that the agent has a limited observation space. It is important to note that the agent is dreaming in a subjective partial observation space, which may also have an effect on learning.

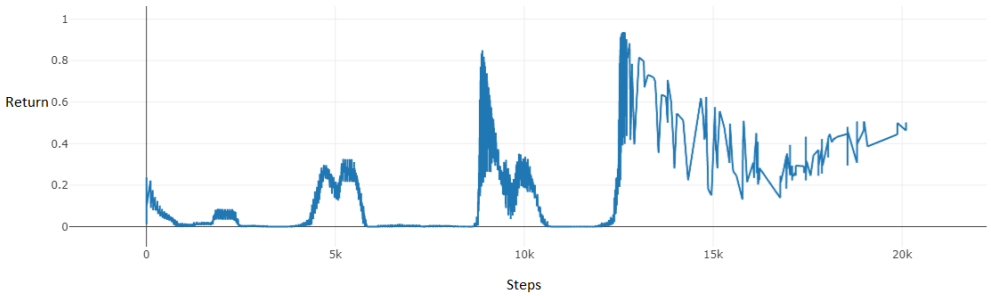


Figure 6: Average Cumulative-return value of the agent while training on the ”MiniGrid-DoorKey-5x5-v0” environment

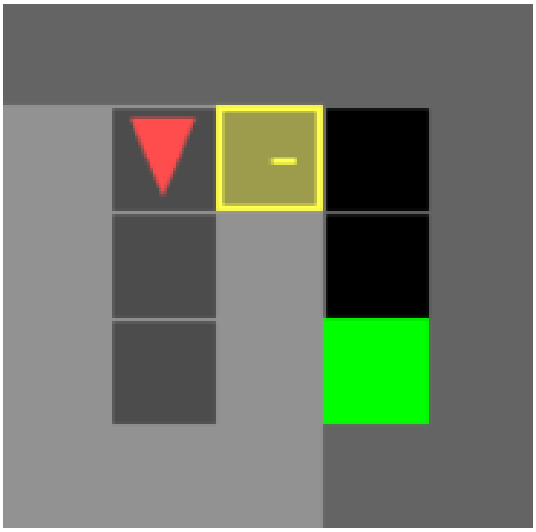


Figure 7: Minigrid DoorKey Env, agent is the red arrow, door is the yellow square, goal is the green square

6.3 Adventure

Adventure is a complex exploration game, with extremely spare rewards, where you have to find a golden chalice in order to get it back to a certain point while dodging a dragon see Fig.8. You get

one negative reward for being killed by the dragon, and a positive reward for finding the chalice, which is hidden in one of 10 different rooms that can be opened through different puzzles, with keys etc...

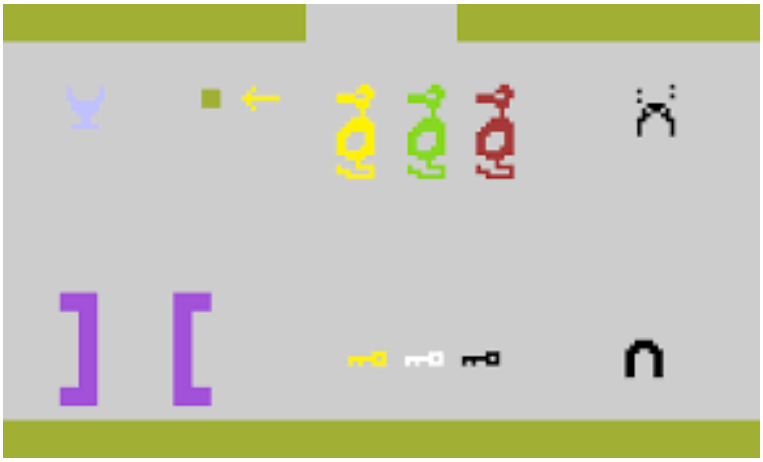


Figure 8: Screen capture of the different elements of the adventure game

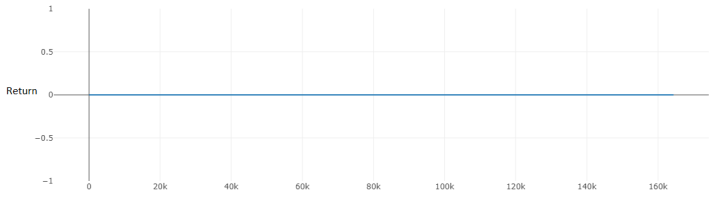


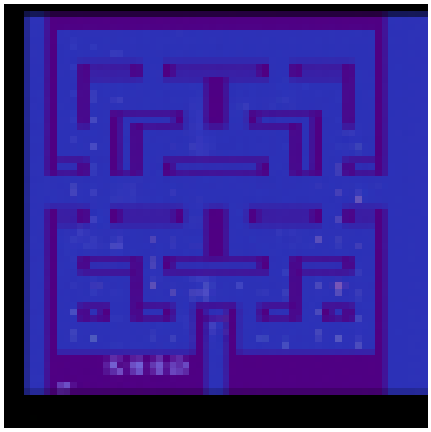
Figure 9: Average Cumulative-return value of the agent while training on the ”Adventure” environment

As you can see in Fig.13, the layout of the world is learned very quickly by the world model. But the agent never managed to get to any rewards, see Fig.9, as they are probably too sparse without additional clues, so the agent wandered around the levels randomly, learning to avoid negative rewards (the dragon) but not much more. As the agent never encountered any rewards, the world model was of no use. The training was stopped after 300,000 steps, as no progress was made. Adventure is really often left out of Atari Benchmarks, as it seems to be still out of reach without hyperparameter tuning or reward engineering.

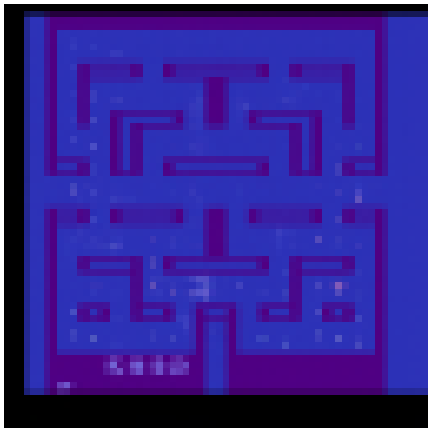
6.4 Alien

This is a Pacman-like game Fig.10 in which you have to collect the white blobs on the ground while avoiding aliens. Here it is interesting to observe the prediction of the trajectories of the enemy aliens, and also the different states of the aliens (edible or not) and the consequence of a player meeting an alien that is edible or not. We can also observe the consequences of a radical level change, as in level 2 of the game the gameplay completely changes.We can also observe the consequences of a radical level change, as in level 2 of the game the gameplay completely changes.

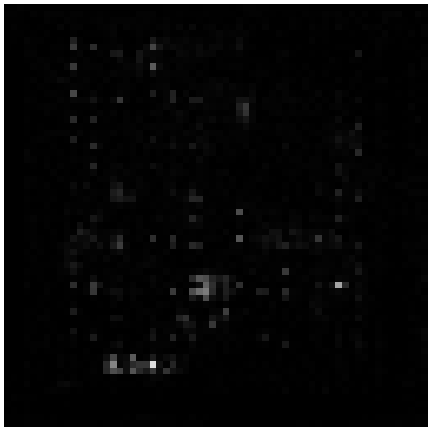
6.5 Dream Interpretation Alien



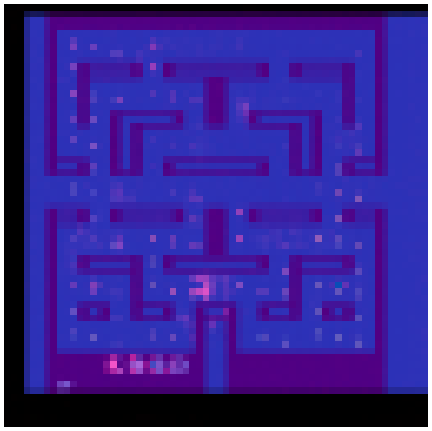
(a) Unmodified dream



(b) Dream with the 8 first lines of the h state set to 1



(c) Difference between (a) and (b)



(d) Superposition of (c) in red on the unmodified image

Figure 10: Influence of the modification of the input h state on the input of the world model

A simple scene with a single enemy has been chosen, and for ease of interpretation the first 8 lines of the z vector have been set to 1. The decoder model is based on the h and z vectors, with $z << h$, so even a significant change in z will not have a completely transformative effect on the output image, as it still has in memory a compression of all the past states that contribute more to the dream. Nevertheless, we can see from Fig.10c and Fig.10d which parts are important for the model to recognise : mainly the **presence of food**, the **position of the enemy**, the **score** and the **presence of the big food**. At the same time, all non-moving features (walls, background, ...) are ignored. It is also noticeable in Fig.10c that the alien moves very little, so the categorical variables have to encode changes in the alien, a kind of velocity, rather than its position.

We can also see in the dreams see Fig.12 and particularly in the gifs [15] impressive feats of prediction where the agent perfectly predicts the position and trajectory of the enemies 50 frames in advance. We can see that at the beginning of the training the algorithm is uncertain about the trajectory of the enemies, so you get a ghosting effect, where the enemy at a branch in a maze in the dream of the future is split in two, in the two plausible paths the alien could take, because the world model is uncertain where it will go. But later in the training it finds a pattern in the alien's movement and is absolutely certain, and that is a huge help to the agent, who can then really confidently avoid the alien.

7 Notable attempts and Further work

In the following section, we will detail the avenues that we did not have time to follow, or that were unfruitful.

7.1 Attempts

A gym environment was created for Sokoban, a game where the emphasis is on moving objects by pushing them sequentially to solve puzzles. This is a common benchmark in RL and planning. The implementation is almost finished, there is just a problem during the training of one of the neural networks, it was dropped because of a lake of time.

There has also been an attempt to look at more advanced techniques for latent variable interpretation, for example using t-SNE doing dimensionality reduction on the latent variables in order to visualise them in 2D, but this approach demonstrated for MNIST in supervised learning in this blog post [18], is difficult to do here where we do not have simple labels that can help users understand the visualisation, but it is still an interesting avenue to explore.

7.2 Further work

The first possible development would be to try more advanced analysis of the network is possible, using interpretable AI techniques(Shapley value, activation maximisation, LRP, ...).

Other interesting problems to look into include real time strategy games(RTS), that require future planning, they could benefit a lot from the dreaming in order to predict good strategies. For example, it would be interesting to try the algorithm on the Lux AI game [19], a ”Polytopia like” turn based, grid based Strategy game as a proof of concept, this game still is an open problem and a competition is being held every year. A natural next step is to try it in the intended setting, in the RAI based game from the LIS laboratory. Another interesting application would be to use this for flying a drone, as this algorithm has already been demonstrated to be capable of robot walk, this could be done in simulation in Gazebo at first as you can't really afford a drone to crash repeatedly in order to learn in the real world, but this could be interesting for mechanics of load balancing when carrying objects, and also maybe for SLAM. These projects should most likely be done using the DreamerV3[14] algorithm, which has shown improved performance across the board and less need for hyperparameter tuning. One last project that would be fascinating is to make an easy-to-use API of the algorithm, especially on dreamer V3 as it is such a powerful and efficient algorithm that needs almost no hyperparameter tuning, giving access to the wider AI community could be really beneficial, this could be done using PyTorchRL [17] which is a new library enabling a high level of modularity in RL.

8 Conclusion

This article has shown that the Dreamer algorithm is capable of making logical predictions for the games it plays, which can greatly help in solving said games. There are still some hidden states that need to be explored, another unknown is also the robustness of the understanding, furthermore this algorithm has not been tested on more complex logical problems. Nevertheless, it shows promising results on a wide variety of problems, which can be easily reproduced with relatively small resources.

The article has also laid the groundwork for games to be easily run on this version of the code, for basic computation techniques to interpret the model, and has shown further points for research with this extremely promising algorithm.

9 Personal takeaways

I have learned a lot in these six months. Firstly, I got to explore the difference between small scale code written on a Jupyter notebook, and research level code. I also saw how quickly published code can become obsolete, as the code from the original paper was really difficult to run. Secondly I learned that as Andrej Karpathy puts it : "the truth lies in the code", the gap between the algorithm explained in the paper, and the efficient way of programming it, is large. And often if you just read the paper you can miss an important detail, looking through the code is the only way I found to truly understand an algorithm. I also have learned a lot about the way I work. Working alone on such a big project has shown me new ways of organizing myself in order to stay productive and motivated, by setting myself small timed objectives, which something I missed at the start of the project. I have also fallen victim to tunnel vision on specific problems, sometimes not realising that they were not important in the overall goal of the project. Finally, I have also understood how hard research can be in contrast to traditional software engineering, as you are often the only person trying to achieve certain things, and problems do not have simple solutions to be found on the internet. I have also now developed a deep love for reinforcement learning, which I can back up with technical skills enabling me to at least understand, if not design, such algorithms.

10 Acknowledgements

Thanks to *Jurgis Pasukonis* for the valuable help on understanding his code and for his insights into the research world in general. Thanks to *Marc Toussaint* and the Learning and Intelligent Systems lab in general for the infrastructure and environment without which this research would not have been possible. Finally, thanks to *Svetlana Levit* for the opportunity to work on such an interesting algorithm,for her patience, and for her great advice.

References

[1] Original Paper, https://arxiv.org/abs/2010.02193, *Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, Jimmy Ba.*

[2] Google AI Blog about dreamer V2,*Danijar Hafner*

[3] The Pytorch repository used to implement the DreamerV2 training protocol Link to Github, *Jurgis Pasukonis*

[4] The original repository written in Tensorflow : Link to the GitHub repository of the original implementation ,*Danijar Hafner*

[5] The Fork of the above repository with the modifications to run additional games, to generate Gifs for additional games, and to evaluate the world models independently. Link to the GitHub repository

[6] Dreamer V1 *Danijar Hafner, Timothy Lillicrap, Jimmy Ba, Mohammad Norouzi*

[7] Day Dreamer Paper in which the dreamer V2 algorithm was applied to the control of a Boston Dynamics spot robot. https://arxiv.org/abs/2206.14176 *Philipp Wu, Alejandro Escontrela, Danijar Hafner, Ken Goldberg, Pieter Abbeel*

[8] PlaNet Blog Post *Danijar Hafner*

[9] Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actors *Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine*

[10] Plan2Explore Paper

[11] Recurrent World Models Facilitate Policy Evolution

[12] Connected Papers Link to see related works

[13] Blog visualization of Recurrent World Models Facilitate Policy Evolution

[14] Link Dreamer V3 Blog with code and paper

[15] Link to google slides presentation where you can see the gifs

[16] Link to the github repository with the code

[17] PyTorchRL Github Page

[18] Latent space visualization

[19] LUX AI Game

11 Annexe

The complete episodes and dreams can be found in the presentation [15]. Following are snapshots of the results.

11.1 Pong

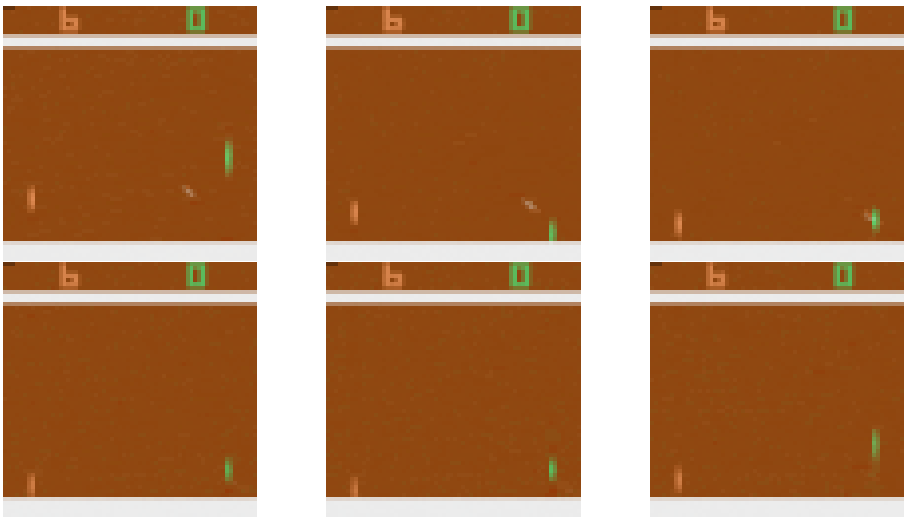


Figure 11: dream of Pong after 10 000 iterations [15], notice the trajectory of the ball is being predicted by the dream, and the rebound trajectory is not yet computed, but noise is starting to emerge along the possible trajectories. See slide 8 of presentation [15].

11.2 Alien

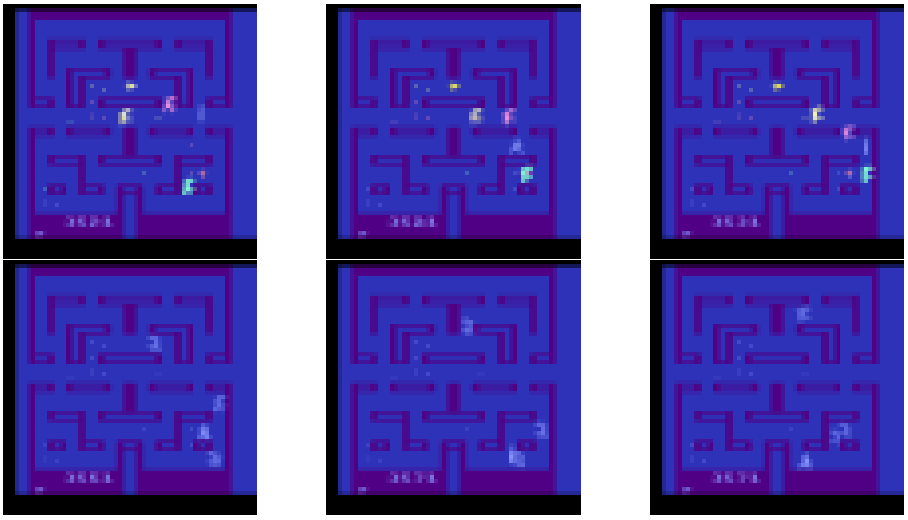


Figure 12: Evaluation of Alien on the after 2 721 000 steps and around 200 000 000 env steps, see slide 11 of the presentation [15].

11.3 Adventure

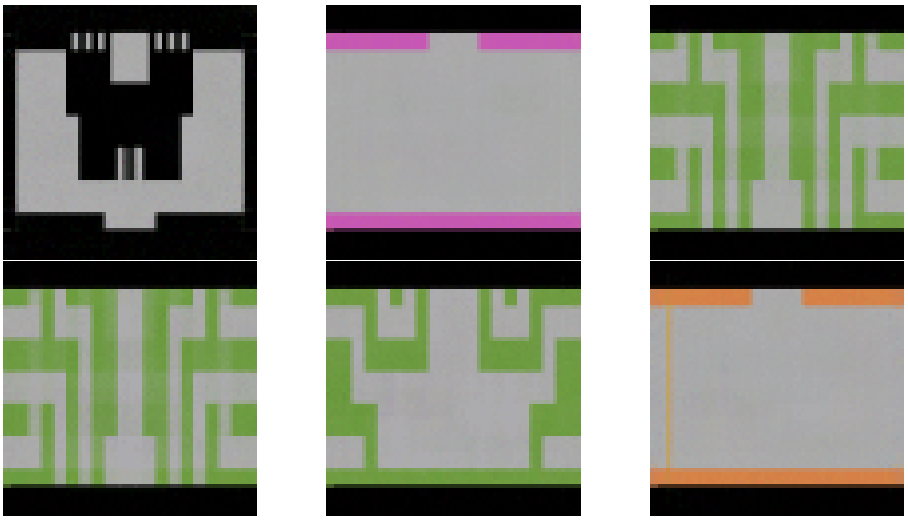


Figure 13: Dream of Adventure [15] after a 1000 step, the algorithm has already learned all the different maps of the game, see slide 10 of the presentation [15].

11.4 Minigrid

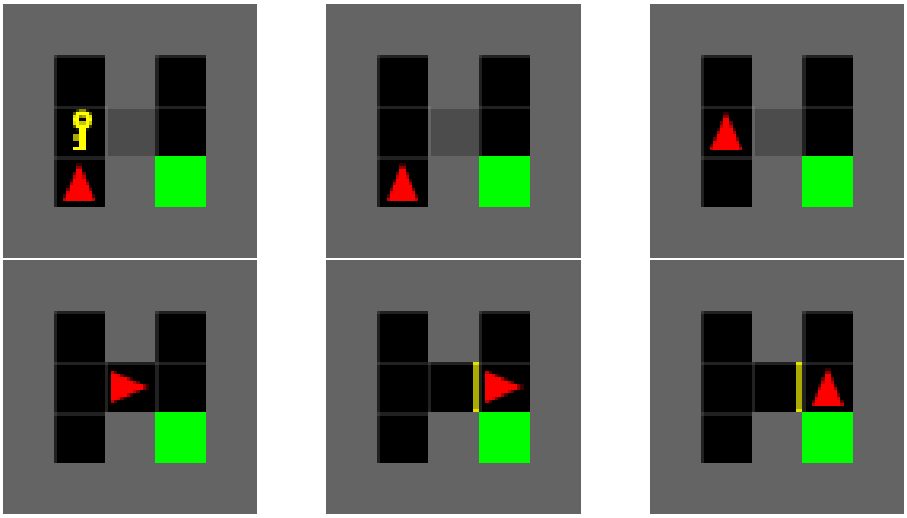


Figure 14: Evaluation episode of Minigrid model trained 83000 steps, see slide 9 of the presentation [15]. The agent is the red arrow, door is the gray square, goal is the green square, the game is being solved successfully.