Planning Goals for Exploration: Pixel-based approach and new Goal-picker Mechanism

Alessandro Marincioni
s3442209

Constantinos Georgiou
s3507637

Abstract

”Planning Goals for Exploration” (PEG) is a method used in Unsupervised Task Discovery where an agent is called upon to perform eligible tasks in an unknown environment. PEG focuses on setting interesting and diverse goals in each training episode with a higher purpose of optimizing an intrinsic exploration reward. In contrast to other Reinforcement Learning domains, an important element in Unsupervised Task Discovery is exploration. PEG uses vector state representation for its environment and two different policies to generate and evaluate goals. We propose pixel-based PEG to extend the algorithm to image domains. In addition, we demonstrate a new goal-picker mechanism in order to achieve faster and more diverse exploration. We show that Pixel-based PEG is a powerful learner both in robotic control and videogame settings. We also demonstrate that our goal-picker mechanism performs better than the default one while having lower computational cost.

1 Introduction

Agents in the Unsupervised Task Discovery domain are expected to detect available tasks in an environment they did not have the chance to interact with in the past. Essentially, this means that during training time, the agent needs to explore the environment to figure out what can be done and what sort of actions should be followed to accomplish each task. Exploration is a crucial element in training since the broader the training, the more generalized the agent becomes. Ideally, this would lead to the accomplishment of new unseen tasks set in evaluation time. Considering this, the goal-conditioned reinforcement learning setting (GCRL) suits better than the classical reinforcement learning one. A natural framework is provided in the GCRL paradigm (Andrychowicz et al., 2017) for training goal-conditioned agent policies on exploration data.

In this work, we build upon Planning Goals for Exploration (PEG), a method that sets distinctive goals for diverse exploration during training under the GCRL setting (Hu et al., 2023). The authors evaluate this method on four distinct environments: Point Maze, Walker, Ant Maze, and 3-Block Stacking. The high-level interpretation of the PEG algorithm is demonstrated in figure 1 with red color. After PEG generates a promising goal, the Go-Explore mechanism is executed so as to reach the goal and explore nearby states. The episode ends by updating the world model, the goal-condition and the exploration policies, and the two value functions. However, PEG is implemented in such a way as to use vector state representation for its environments and goals. Intuitively, this approach saves precious time during training by encoding only the essential and most informative data that appear in each environment. Another thing to consider is the various elements needed for generating a single goal. As mentioned, it is important to update five elements to generate a novel goal for the next episode.

We hypothesized that the state representations could be encoded with pixels instead of vectors. Although Pixel-based representations can have high-dimensional input spaces, they are able to capture fine-grained details of each environment. Moreover, a pixel goal representation allows for a more natural way of representing goals and is closer to the one humans use. Apart from that, in the present work, we propose a new goal-picker mechanism that is based on the Novelty Search algorithm. In particular, the novelty part is enclosed in an evolutionary algorithm to output an interesting as well as unordinary goal and avoid leveraging any of the five elements. We conjecture that this substitution in the goal-generation phase benefits not only the quality of the algorithm but also its speed.

Our experiments show that PEG models can be trained effectively using pixel inputs, both in robotic control tasks and in videogame settings. Working in the image space we qualitatively and quantitatively show the strengths and weaknesses of the PEG algorithm and suggest possible ways to improve it. We also present a new goal-picker mechanism for the PEG algorithm, based on the Novelty Search algorithm. The performance of the new mechanism is evaluated in terms of computational cost, performance, and robustness to hyperparameter changes.

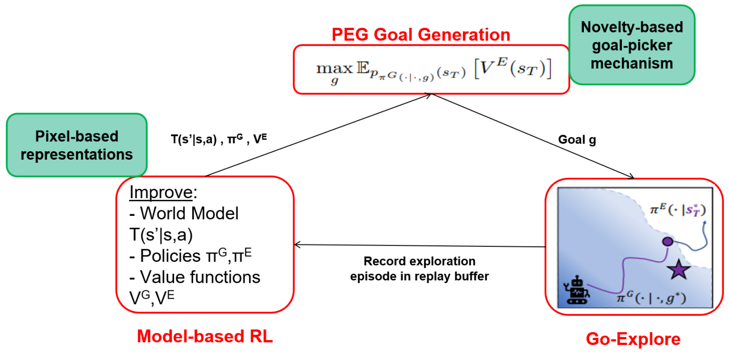
PEG Goal Generation
$\max_g \mathbb{E}_{p_{\pi^G(\cdot|\cdot,g)}(s_T)}\left[V^E(s_T)\right]$
Novelty-based goal-picker mechanism
Pixel-based representations
T(s'|s,a) , π^G , V^E
Goal g
Improve:
- World Model T(s'|s,a)
- Policies π^G,π^E
- Value functions V^G,V^E
Record exploration episode in replay buffer
$\pi^E(\cdot\,|s_T^*)$
$\pi^G(\cdot\,|\,\cdot, g^*)$
Model-based RL
Go-Explore

Figure 1: PEG Training Loop. Each training episode is divided into three main steps. Firstly, PEG finds the goal that leads to the highest exploration reward. The next step is to apply the Go-Explore mechanism with the chosen goal g. Finally, update the world model, the two policies, and the two value functions. The two green frames indicate the improvements in our work. Use Pixel-based representations of the environment instead of Vector-based ones. In addition, a new goal-picker mechanism is introduced based on the Novelty Search Algorithm.

2 Background

2.1 Goal-Conditioned RL setting

We consider a goal-conditioned Markov decision process, defined by the tuple (S, A, T, G). A indicates the action space, while S and G are the state and the goal spaces, respectively. Our objective is to build the following policy $\pi^G(a_t|s_t, g)$. Essentially, for each timestep t, it considers the current state of the agent $s \in S$ and selects an action $a \in A$ under a goal command $g \in G$. It then transitions to the next state s' with a probability given from the dynamics matrix T. We set the goal space to be identical to the state space: $G = S$, that is, every state maps to a plausible goal-reaching command. In contrast to other works (Liu et al., 2022), we do not take into consideration the reward function and the test-time goal distribution is unknown to the agent.

2.2 Go-Explore mechanism

We ensue the "Go-Explore" mechanism as it is described in the paper "First return, then explore" (Ecoffet et al., 2021). This means that we maintain two policies: a goal-conditioned policy π^G and an exploration policy π^E. For each training episode, we start with the "Go phase", where we try to approach a goal g following the goal-conditioned policy π^G. After T_G steps, we arrive at a terminal state s_T and we switch our policy to the exploration policy π^E to explore nearby states for the remaining T_E steps. An exploration policy can be built on top of the Reinforcement Learning framework such as $\pi^E(a_t|s_t)$. In PEG, the exploration policy π^E was trained to maximize an intrinsic exploration reward r^E. It indicates how novel an ending state is if we follow the exploration policy starting from the terminal state s_T and predicting every next state using a world model T.

3 Related Work

3.1 Unsupervised Goal-Conditioned RL

”Latent Explorer Achiever” (LEXA) (Mendonca et al., 2021) is a similar algorithm for discovering new goals and reliably achieving them. PEG and LEXA seem to share a common ground when it comes to the structural aspect since both of them leverage a world model, an explorer, and an achiever component. Although PEG uses vector state representations for efficient training time, authors of LEXA have shown that working on pixel inputs allows having a more enriched representation of the environment. This, in turn, enables more interesting applications, such as training a model to solve multiple different tasks.

3.2 Novelty Search

Novelty Search is an algorithm for finding unusual behaviors (Stanley & Lehman, 2015). Many works tried to imitate this idea. An appealing example can be seen in the paper ”Revising the Evolutionary Computation Abstraction: Minimal Criteria Novelty Search” (Lehman & Stanley, 2010). The authors try different approaches to this principle to solve two maze navigation environments. Some of the key aspects of this paper are the following: NEAT (NeuroEvolution of Augmenting Topologies), Novelty Search and Minimal Criteria Novelty Search.

4 Methodology

4.1 Pixel-based PEG

4.1.1 Algorithmic Description

The first contribution of this work is to use pixel-based representations instead of vector-based ones. To adapt PEG to work with pixel-based representations we need to make some changes to the algorithm. First of all, the world model, the goal-conditioned policy, and the exploration policy are all trained on state embeddings. These embeddings are generated by a neural network that takes a state as input. The embedding network was changed to a convolutional neural network to adapt to pixel state representations.

MPPI, the goal generation mechanism of PEG, also had to be changed. MPPI is designed to generate vector goals by fitting a Gaussian distribution. This method is applicable to vector goals because of the high density of meaningful states in vector space. On the contrary, pixel space is very sparse thus the goal-picking strategy was changed to randomly sample a batch of goals from the replay buffer and pick the one with the highest exploratory value. Figure 2 shows an outline of the goal-picking mechanism.

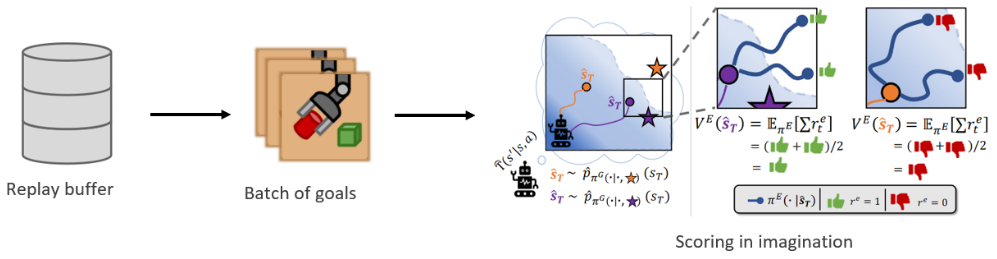
Replay buffer
Batch of goals
$\hat{s}_T$
$\hat{s}_T$
$\hat{T}(s'|s,a)$
$\hat{s}_T \sim \hat{p}_{\pi^G(\cdot|\cdot,\star)}(s_T)$
$\hat{s}_T \sim \hat{p}_{\pi^G(\cdot|\cdot,\star)}(s_T)$
$V^E(\hat{s}_T) = \mathbb{E}_{\pi^E}[\sum r_t^e]$
$= (👍 + 👍)/2$
$= 👍$
$V^E(\hat{s}_T) = \mathbb{E}_{\pi^E}[\sum r_t^e]$
$= (👎 + 👎)/2$
$= 👎$
$\pi^E(\cdot|\hat{s}_T)$
👍 $r^e = 1$
👎 $r^e = 0$
Scoring in imagination

Figure 2: Goal picking strategy for pixel PEG. A batch of goals is sampled from the replay buffer, and each goal is scored on its expected exploratory value in imagination using the world model. The best-scoring goal is then used for training.

4.1.2 Experimental Design

PEG was evaluated on two different environments: 3-Block Stacking and NES Super Mario Bros.

The first experiment focuses on comparing the performance of pixel PEG, pixel LEXA and vector PEG in the 3-Block Stacking environment. The environment was chosen due to it being the most challenging out of the environments presented in the original PEG paper. Pixel PEG and LEXA are trained for 6M steps, whereas vector PEG is trained for 1M. The average success rate on the set of tasks of the 3-Block Stacking environment is used for comparison. It is important to note that the goals between the pixel and vector environments differ slightly since images of the goal state also contain information about the robotic arm.Experimental results show that most of the model uncertainty lies in block dynamics, thus this difference is considered negligible.

The performance of pixel PEG was also evaluated in the NES Super Mario Bros environment. A pre-existing environment was adapted to the GCRL setting for this purpose. 11 goal states were set for the environment by selecting progressively more advanced positions in the game levels and saving the game rendering. PEG is trained for 10M steps on the environment and evaluated both qualitatively and quantitatively on the average success rate. This experiment also features a longer episode horizon of 150 steps instead of the default 50 to adapt the agent to the videogame levels, which take significantly more time to complete than control tasks, which PEG was originally developed for. Frame skipping is also used, with one every 4 frames being provided to the agent.

The hyperparameters for each task had to be adapted from the default vector PEG settings to take memory constraints into account and adapt to the new tasks. A new configuration setting is defined for each image task, exact changes can be found in the github repository.PEG was also tested on a simple 2D maze environment but did not show any signs of learning, so no further experiments on that environment were conducted. These results and a brief discussion can be found in Appendix A.1.

4.2 Goal-picker Mechanism

4.2.1 Algorithmic Description

For this contribution, we are focused on how we can pick novel goals. We introduce an alternative way that is simpler for the first step ”PEG Goal Generation” illustrated in figure 1. In particular, we apply the Novelty Search algorithm with the idea of rewarding instances whose functionality is drastically different from what has been discovered previously. Note that we are interested in the Point Maze environment for the current contribution. First of all, the novelty score of a solution is computed by taking into consideration the behavior of other solutions. We aim to measure sparseness and promote the solution whose behavior differs the most. The term ”behavior” is relatively general to suit any kind of environment and in our work behavior indicates a two-dimensional point since our maze is a two-dimensional grid. In addition, k-nearest neighbors was chosen to efficiently measure and evaluate the sparseness of a point. Therefore, the novelty score of a point x is given below:

$$\rho(x) = \frac{1}{k}\sum_{i=0}^{k} dist(x, \mu_i)$$

where k indicates the number of neighbors considered and μ_i represents the behavior of the i^{th} neighbor, that is a two-dimensional point. The higher the score, the more novel the solution is. In our case, we integrate the novelty part into an Evolutionary Algorithm since we have an optimization problem; meaning we want to produce more and more novel goals as we proceed with training episodes. The pseudocode below describes our approach which is applied in every training episode.

The evolutionary part can be seen in lines 5-10. For every iteration, we first generate $|X|$ random points in the generation scope. Each point represents a different state that can be chosen and, thus, we evaluate its novelty. Simply, for every generated point, we find its k-nearest points or goals that the agent has already visited and we assign its novelty score. Before moving on to the next generation, we reduce the generation scope. The generation scope designates the space in which we are allowed to generate points in order to imitate the exploration-exploitation trade-off. At the beginning of each training episode, the generation scope covers the full two-dimensional maze to promote exploration. As we iterate through generations, we make the scope smaller to force the agent to produce points in areas that are considered to be novel. Lines 1-3 help us tackle the cold-start problem where the agent has not visited any goal states yet and its goal buffer is empty. Regarding the goal buffer, there are two things we can do when we meet the buffer limit: either we forget all the states we visited

Algorithm 1 Evolutionary Algorithm with Novelty Search

```
1: if $buffer$ is Empty then
2:     Generate a random point.
3:     Add the point to the $buffer$.
4: else
5:     for $generation = 1, 2, \ldots$ do
6:         Generate $|X|$ random points.
7:         Evaluate all solutions using the Novelty score as the fitness function.
8:         Pick the goal with the maximum Novelty score.
9:         Reduce the generation scope.
10:    end for
11:    Add the chosen goal from the last generation to the $buffer$.
```

so far or we remove the oldest one. Please refer to Figure 3 for visualizing the pseudocode for a random training episode.

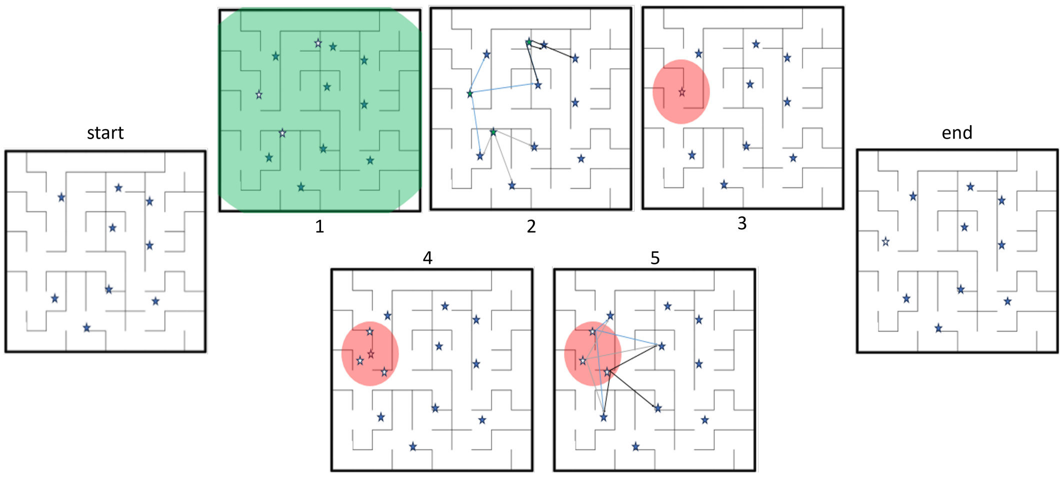

Figure 3: Visualization of our Goal-Picker Mechanism for a random training episode. 1) Generate $|X|$ random points in the generation scope which covers the whole maze. 2) For every generated point, find the k-nearest goals that the agent has already visited and compute the novelty score. 3) Reduce the generation scope. 4+5) Repeat the same procedure for the next generation.

4.2.2 Experimental Design

To conduct our experiments, we ran the agent for 600 thousand training episodes using the Point Maze environment. The maze is a two-dimensional grid consisting of blocks and eligible positions. At any time, the agent can be in a position $(x1, x2)$, where $x1$ and $x2$ represent the dimensions and can take real numbers from 0 to 10. We also set a goal state $(x1_g, x2_g)$ which we change at every training episode to force the agent to learn the environment and the possible actions at every state.

To evaluate our goal-picker mechanism, we tried different values for the hyperparameters of the algorithm. Regarding the goal buffer, we tried different sizes such as 1000, 10.000, and 100.000. Moreover, when we met the maximum size for the goal buffer we either forgot all the states we visited so far or we removed the oldest one. Two of the most important hyperparameters are the number of generated points and the number of neighbors considered to compute the novelty score. For both hyperparameters, we experimented with the values 3,5 and 10. The generation scope is reduced gradually depending on the number of generations. For example, if we set the number of generations to be equal to 2, then in the 1^{st} generation the scope will cover the full maze and in the 2^{nd} generation it will be reduced by half. Lastly, we experimented with the following values to represent the number of generations: 1,3 and 10. We assess these variations with the PEG goal generation mechanisms MPPI (the default one) and MEGA.

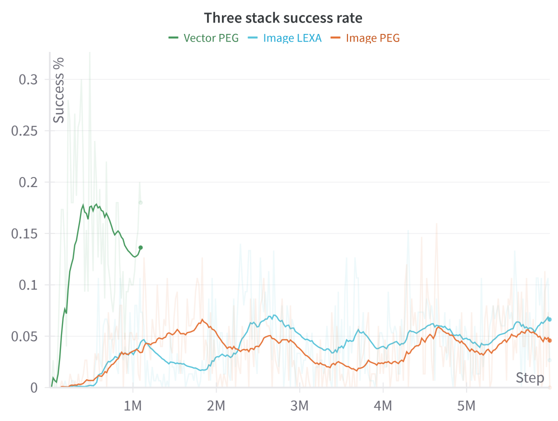

Figure 4: Mean success rate of the 3-Block Stack environment agents. Vector PEG performs the best, whereas image PEG and LEXA perform similarly. Both image models have fluctuating performance during training.

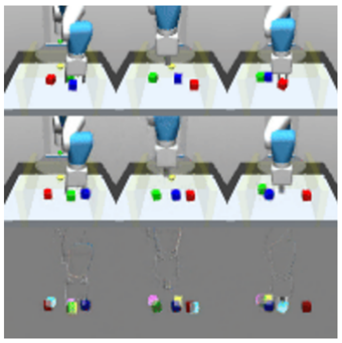

Figure 5: Predictive error of the world model at the end of an episode. Rows from top to bottom represent the ground truth dynamics, predicted dynamics and prediction error. Most of the prediction error concerns the position of the blocks.

5 Results

5.1 Pixel-based PEG

5.1.1 3-Block Stack environment

The performance of the three algorithms on the 3-Block Stacking environment is shown in figure 4.

The results show that pixel PEG and LEXA perform similarly, performing worse than vector PEG. This is expected since working with pixel inputs is more challenging than working with vector inputs, which encode rich information about the environment. The results also show that both pixel PEG and LEXA have fluctuating performance, but seem to stabilize on similar values over time. Figure 5 shows the representation error of the pixel embeddings after 1M steps of training. The results show that most of the predictive error of the world model is in the dynamics of the blocks, which is expected due to the complexity of modeling their interactions with accuracy from pixel data. This supports the validity of our comparison between pixel and vector PEG, showing that controlling the robotic arm is an easy task and does not impact the performance of the agent.

5.1.2 NES Super Mario Bros

The mean success rate for the pixel PEG agent on this task is shown in Figure 7. The results show that the agent performance fluctuates significantly during training. The low success average success rate is mostly impacted by the difficulty of the later goals. These goals are usually too far from the starting position, and the agent is not able to reach them in the episode horizon. Figure 6 shows the final state achieved by the agent for each goal. The results show that the agent is able to the easier goals successfully. In these goals, smaller details like the exact position of Mario and its power-up state are often ignored, since they make up a small part of the image state. However, the agent learns to control the character to scroll the screen to the right, to line up with the goal position correctly. For harder goals, the agent is not able to reach the goal position in the episode horizon, but often attempts to reach it by exploring the level to the right which is a promising sign. Observations on recorded training episodes show that the chosen goals are states that are far to the right of the starting position. The explorer policy also focuses on exploring the level to the right, where the agent is able to find new unseen states. Video results showcasing this and other behaviors can be found in the project repository [1].

[1] `https://maranc98.github.io/image_peg_experiments/`

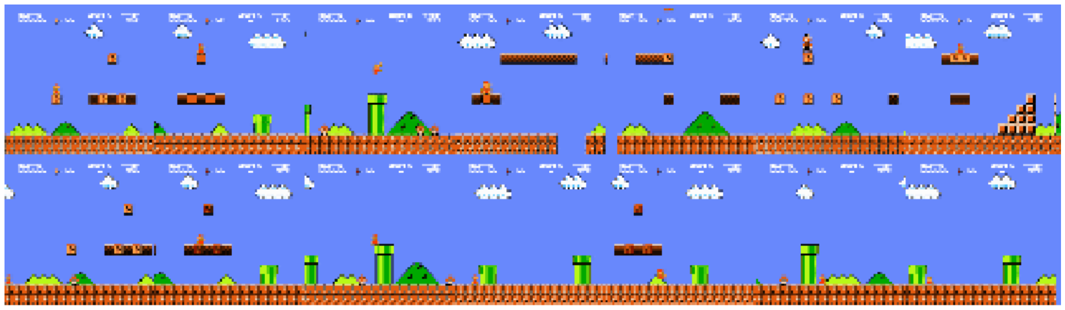

Figure 6: Final state achieved by the trained PEG agent on the NES Super Mario Bros environment. Top row shows a set of goals, bottom row shows the corresponding final state that was achieved. The agent learns to move in the level to align itself with the goal state. Finer details like the power up state of Mario or the state of enemies and blocks are ignored by the model.

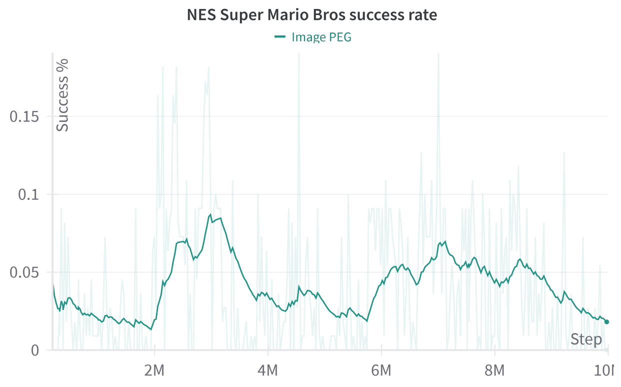

Figure 7: Mean success rate of pixel PEG on the NES Super Mario Bros environment. While the agent is qualitatively observed to learn interesting behaviors over time, the mean success rate curve is not always increasing. This could be due to the fact that when focusing on new tasks the model is prone to forgetting previously learned behaviours. Note that the success rate underestimates the performance of the agent as some goals are not actually achievable in the episode horizon.

5.2 Goal-picker Mechanism

The results of this contribution are divided into two different types: quantitative and qualitative evaluation. The former one can be seen in figure 8. On the left-hand sight, we can see four lines: the PEG goal generation method which represents the baseline, and three variations of our algorithm. We chose the specific three variations since they demonstrate slightly better performance than the baseline. Our algorithm with two generations (cyan line) is really close to the baseline, while with one generation (purple line) it achieves maximum success in less than 300 thousand training episodes, with no other approach being faster than that. On the right-hand sight, we compare our best variation with the PEG's default approach. The first observation we can make is that our approach learns as fast as PEG's. The second insight is that our approach achieves better performance after 600 thousand steps; about 7% more success rate!

All results concerning the qualitative evaluation are illustrated in figure 9. We can see snapshots of the agent (red dot) for eight random training episodes. The blue dot shows the target position which is modified at every training step. A first observation is that our method sets more diverse goals than PEG. For example, the chosen goals by PEG for the 370, 520, and 590 thousand episodes are in the same area (bottom left corner). By collecting many snapshots we were able to observe how the agent was acting at the corresponding training episodes. In conclusion, our agent gets closer to the goals at each training episode, which in turn reflects the quantitative evaluation and the fact that our mechanism is in general more successful.

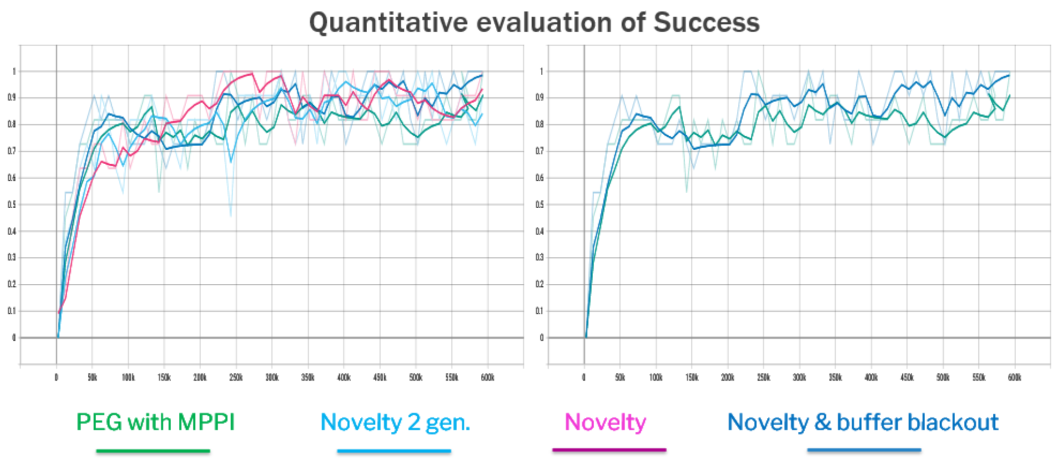

Figure 8: Quantitative Evaluation. We compare three variations of our algorithm with PEG's default goal-generation procedure for 600 thousand training episodes. All three variations seem to be at least as good as the baseline. The faster way to get the maximum success rate is with just one generation (purple line). On the right-hand sight, we compare our best variation with the PEG's default mechanism. After 600 thousand training episodes, our algorithm achieves a 98% success rate which is higher than PEG's mechanism by 7%.

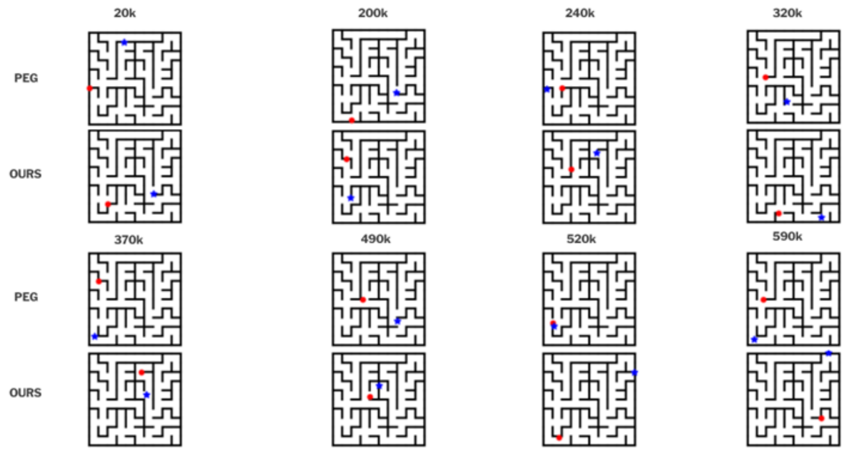

Figure 9: Qualitative Evaluation. We collected snapshots of the agent using the PEG's goal generation approach and also our mechanism for eight different timesteps. Our algorithm produces more novel goals episode by episode. By collecting many snapshots, we saw that when the agent used our algorithm it got closer to the target goal.

6 Discussion

Our experiments on the 3-Block Stacking environment have shown that using pixel inputs is more challenging than using vector inputs. Pixel PEG and LEXA are capable of learning to act in the environment to some extent, but they are not able to reach the performance of vector PEG. This is expected since vector inputs encode rich information about the environment, specifically the pose of the three blocks. In addition, pixel PEG and LEXA have shown fluctuating performance. This behavior of forgetting old skills has also been observed in the original PEG paper and is a known issue in the GCRL setting. Results on the average success rate of the models seem to show that the Go Explore mechanism and the simple goal generation mechanism are not enough to meaningfully set PEG apart from LEXA in the pixel version of the 3-Block Stacking environment. Ablation studies on the vector PEG algorithm have also shown that training the model on seen goals, i.e. using

the goal-picking strategy implemented in pixel PEG, does not reduce the performance of the agent significantly. Despite all these, the performance gap observed between vector PEG and pixel PEG is no longer present. A possible explanation for this is that being able to correctly model the dynamics of the blocks is fundamental to the task, and since no model is capable of doing so sufficiently well from pixel inputs, they both incur the same performance limitations. More experiments on this environment are needed to confirm this hypothesis.
As for the NES Super Mario Bros environment, the results show that pixel PEG can learn to control the character to reach the most feasible goals in the environment. The videogame environment also allows us to better visualize the behaviour of the agent, and some interesting exploratory behaviors have been observed. One limitation of the PEG algorithm for video game settings is its limited episode horizon. The agent is not able to reach goals that are too far from the starting position, and this limits the complexity of the tasks that can be solved, even after implementing frame skipping and using an increased episode horizon. Moreover, using longer episode horizons also increases the computational cost of the algorithm, since each episode used for training takes longer to complete. The whole trajectory of the agent is also stored in the replay buffer, which increases the memory requirements of the algorithm.
Regarding the proposed goal-picker mechanism, we observed that its performance is highly dependent on the hyperparameters of the algorithm. Although in this article we present results of our mechanism with at least as good performance as PEG's default mechanism, we saw fluctuating and poor performance in other combinations of hyperparameters. Another observation is that despite the fact that our mechanism is based on mathematical calculations and no neural networks are used, our mechanism is faster only by 5%. To conclude, it is worth mentioning that some of the hyperparameters help our mechanism achieve faster near 100% success, whereas others make our mechanism stand out in the long run of 600 thousand training steps with 7% higher accuracy than PEG's default one.

7 Conclusion

In this work, we proposed two different contributions to the PEG algorithm. The first one is to use pixel-based representations. Our hypothesis was that pixel-based representations can capture fine-grained details of each environment. We evaluated our approach in two environments. Pixel PEG can learn to control the robotic arm to reach some goals in the 3-Block Stacking environment, but it cannot reach the performance of vector PEG, likely due to the difficulty of modeling the dynamics of the blocks from pixel inputs. We have also shown that pixel PEG can learn to control the character in the NES Super Mario Bros environment to reach the most feasible goals. Limitations of the algorithm have also been discovered, such as the constraints caused by the episode horizon, which prevents the agent from reaching goals that are too far from the starting position making PEG unsuitable for long-term goals. The second contribution of this work is a new goal-picker mechanism for the PEG algorithm. We have proposed to use the Novelty Search, which was tested on the maze environment. We show that our mechanism is faster and sometimes even better. The efficiency of the algorithm is highly impacted by the combination of hyperparameters of the algorithm such as goal buffer size, number of generations, neighbors considered, size of generation scope and many more.

There is plenty of room for future work on this project. Firstly, a more thorough evaluation of pixel PEG on the 3-Block Stacking environment is needed to confirm the hypothesis that the performance gap between pixel PEG and vector PEG is due to the difficulty of modeling the dynamics of the blocks from pixel inputs. It would also be interesting to experiment with different goal-generation strategies for pixel PEG. The goal-picking strategy implemented in pixel PEG is very simple, and it would be interesting to see if more complex strategies, such as generative models like VAEs, GANs, or diffusion models, could improve the performance of the algorithm. Another interesting direction for future works is to experiment with longer episode horizons. A long-horizon version of the PEG algorithm would allow one to employ the algorithm on more complex tasks, with goals that require many steps to be reached. This would also allow to use the algorithm in not only video games but also real-world environments which require many steps to complete a task.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 2017.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, pp. 580–586, 2021.

Edward S. Hu, Richard Chang, Oleh Rybkin, and Dinesh Jayaraman. Planning goals for exploration, 2023.

Joel Lehman and Kenneth O Stanley. Revising the evolutionary computation abstraction: minimal criteria novelty search. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, 2010.

Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299*, 2022.

Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, pp. 24379–24391, 2021.

Kenneth O Stanley and Joel Lehman. *Why greatness cannot be planned: The myth of the objective*. Springer, 2015.

A Appendix

A.1 Pixel PEG on 2D Point Maze

Preliminary experiments were conducted on the 2D Point Maze environment. In this environment the agent can move horizontally and vertically and is tasked with reaching a target destination.

Figure 10 shows the final state that is achieved by the pixel PEG agents after training for 1M steps. What is observed is that throughout training, the agent never learns to leave areas close to the starting bottom left position. This behaviour is quickly learned, but no progress is made throughout the 1M steps of training. Figure 11 shows the world model predictions on this environment. It shows that the world model is capable of correctly modeling the environment, since all necessary information to solve the task is present in the world model predictions.

The behaviour of the trained model suggests that the PEG paradigm is incapable of finding good goals for the agent. The agent is trained on goals with high epistemic uncertainty, meaning that an ensemble of models has high disagreement on what the outcome of a given action in that state would be. Due to the simplicity of the environment it is indeed possible that states far from the starting position are correctly modelled very quickly, meaning that the agent has no interest on picking goals, and thus training, on goals that are farther from the starting position. Since these goals are also sparser, as the agent is more likely to be close to the origin when walking randomly, picked goals do not push the agent to explore most of the maze and learn.
This behaviour is not observed in the vector representation of the environment, as agent observations are only its current position. This means that the agent needs to discover which paths are open and which paths are not. Thus when walking on unexplored territory, the ensemble of models has to guess whether a wall is present or not, causing high uncertainty in unexplored areas.

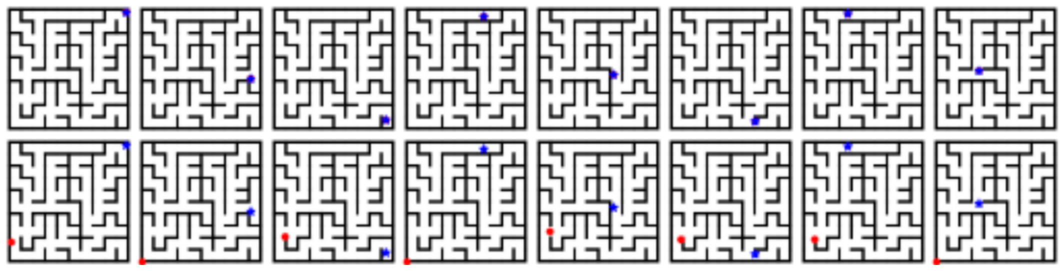

Figure 10: Final state of the pixel PEG agent when given a set of goals on the 2D point maze environment. The agent does not learn to explore far from the starting position (bottom left).

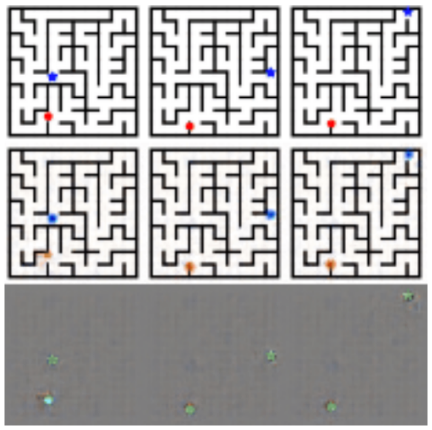

Figure 11: World model predictions on the 2D point maze environment. Rows top to bottom show the ground truth, world model prediction and error respectively. The world model is able to encode all the necessary information to reconstruct and predict the evolution of the environment.