A Environment Details

A.1 Maze

Environment. The mazes that we consider in this paper are implemented as MiniGrid environments [82]. Each maze is a 16×16 grid containing walls and empty cells, two of which are the starting and the goal cells. An agent solving the maze starts at the starting cell and observes a 5×5 area around itself. The agent can move forward into an empty cell or turn left or right in its own cell. To maintain consistency with the MiniGrid environments, the agent is also allowed to pick up, drop or toggle an object or notify that a task is done. In the mazes generated by our work, all those actions result in the agent staying in the same cell. A time limit of 648 is used since an optimal agent will be able to finish all possible 16×16 mazes in this duration. If the agent reaches the goal within this time limit, it receives a reward of $1 - 0.9\times$ fraction of the time limit used. Otherwise, the agent receives no reward.

Environment generator. The environment generator accepts a 16×16 bit map denoting the walls and empty spaces as the input. For better visualization, we add a wall surrounding the 16×16 region. We set the starting cell and goal cell to be the pair of empty cells that are furthest apart, as identified by the Floyd-Warshall algorithm [93].

Agent. We select an agent from a recent work on open-ended learning, ACCEL [4], for the purpose of evaluation. Since individual ACCEL agents have a high variance in their performance, we evaluated the agents trained with four different random seeds on three of the test mazes given in the original paper (*Labyrinth*, *16Rooms*, *LargeCorridor*). We chose the best performing agent out of the four and fixed it for all our experiments. The selected agent was able to always reach the goal in those test mazes.

A.2 Mario

Environment The Mario environments that we consider in this paper are implemented in the Mario AI Framework [94, 21]. Each level is a 16×56 grid of tiles, where each tile can be one of 17 different objects. The agent in each environment receives as input the current game state, consisting of all tiles that are visible on the screen. The agent then outputs the action for Mario to take. Each episode runs for 20 time ticks.

Environment generator. Drawing from prior work [16, 58], the Mario environments are generated with a GAN pre-trained on human-authored levels with the WGAN algorithm [95, 96]. The GAN's generator takes as input a latent vector of size 32 and outputs a 16×56 level padded to 64×64. The GAN architecture is shown in Fig. 5.

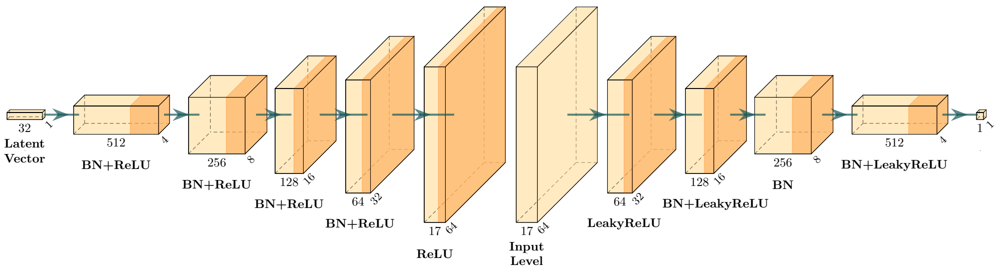

Figure 5: GAN architecture for generating Mario levels (BN stands for Batch Normalization [97]).

Agent. In each environment, we run the A* agent developed by Robin Baumgarten [22]. This agent won the Mario AI competitions at the ICE-GIC conference and the IEEE Computational Intelligence in Games symposium in 2009. The trajectory taken by the agent in a level is stochastic due to randomness in the environment dynamics.

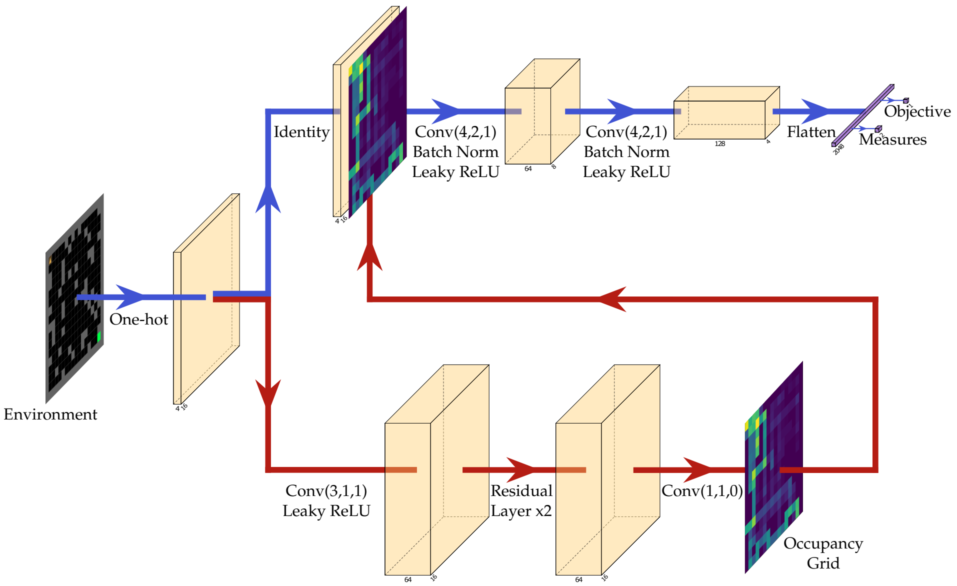

Figure 6: Architecture of the surrogate model. The model predicts the occupancy grid (red arrows) which guides the downstream prediction of the objective and the measure values (blue arrows).

B Deep Surrogate Model

In the DSAGE algorithm, we maintain a deep surrogate model (Fig. 6) for predicting the objective and the measures resulting from simulating an agent's execution in the environment. The input to this model, provided as a one-hot encoded representation of the image of the environment, is passed through a two-stage deep surrogate model as described in Sec. 4.

The first stage predicts the ancillary agent behavior data that is in the form of an occupancy grid. The predictor consists of a 3×3 convolution (with Leaky ReLU activation) followed by two residual layers [98] and a 1×1 convolution. Since the occupancy grid depends on the layout of the environment, we believe that residual layers' propagation of the input information is helpful for prediction.

The predicted occupancy grid and the one-hot encoded image of the environment are stacked and passed through another CNN that predicts the objective and the measure values. The architecture of this CNN is inspired by the discriminator architecture in prior work on generating Mario levels with a GAN [16, 58]. The input is passed through layers of 4×4 strided convolutions with a stride of 2 and an increasing number of channels. Each convolution is followed by Batch Normalization [97] and LeakyReLU activation. Once the height and width of the output of a convolution have been reduced to 4, it is flattened and passed through two fully connected layers to obtain the objective and the measure values.

DSAGE Basic and DSAGE-Only Down do not predict the occupancy grid. The surrogate model in those algorithms directly predicts the objective and the measure values as denoted by the blue arrows.

B.1 Evaluating the Prediction Performance

Mean absolute error. To test the prediction performance of the deep surrogate model trained by DSAGE and its variants, we select two separate runs of each algorithm. The datasets generated in the first run of each algorithm are combined into a single dataset. We then evaluate the trained surrogate models from the second run of each algorithm on the combined dataset by calculating the mean absolute error (MAE) between the predicted and the true objective and measures corresponding to the solutions in the combined dataset.

Table 2 shows the obtained MAEs in the Maze and the Mario domains. In both domains, we observe that the measures that depend on agent behavior (mean agent path length for Maze and number of jumps for Mario) are harder to predict compared to the ones that only depend on the environment

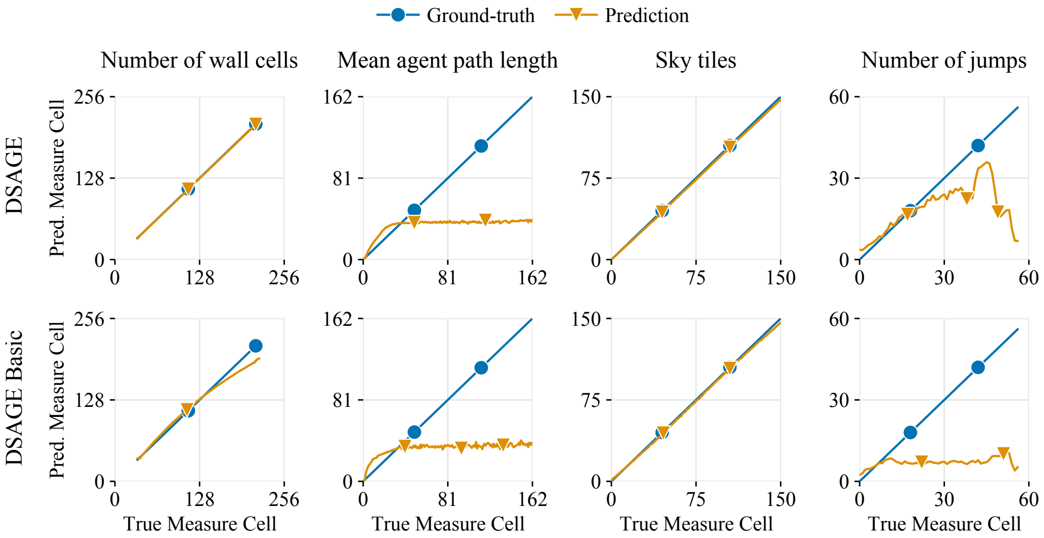

Figure 7: Correlation between the predicted and the true measure cells. The first row corresponds to DSAGE while the second row corresponds to DSAGE Basic. The columns correspond to the two measures in the Maze and the Mario domains respectively. We observe that long agent path lengths and high numbers of jumps are more difficult to predict.

(number of wall cells for Maze and number of sky tiles for Mario). Indeed, the MAEs for the number of wall cells in the Maze domain and the number of sky tiles in the Mario domain are much smaller than the MAE for the mean agent path length and the number of jumps, respectively.

In the Maze domain, predicting ancillary agent behavior data helped improve the prediction of the mean agent path length. Both DSAGE and DSAGE-Only Anc have better predictions compared to their counterparts that do not predict ancillary data. Since the mean agent path length is a scaled version of the sum of the occupancy grid, having a good prediction of the occupancy grid makes the downstream prediction task much easier. We believe that the additional supervision during training in the form of the occupancy grid guides the surrogate model towards understanding the layout of the maze and the agent's behavior.

On the other hand, we see little improvement when predicting the number of jumps in the Mario domain. Here, downsampling provided a larger boost to the predictions, with DSAGE and DSAGE-Only Down making better predictions than their counterparts without downsampling. Since we do not store temporal information in the occupancy grid, predicting the number of jumps remains a challenging task even with an accurate prediction of the occupancy grid. We conjecture that the increased number of outer iterations when downsampling played a more important role in correcting the errors of the surrogate model and improving its predictions.

Correlation plots. To further test if DSAGE's predictions of some measures were more accurate in certain regions of the archive, for each solution we plot the true measure cell on the x-axis and the average of the corresponding predicted measure cell on the y-axis (Fig. 7). In this plot, accurate predictions would fall on the $x = y$ line (denoted in blue), and inaccurate ones would be above or below the line.

Once again, we see that the measures dependent on agent simulation, i.e., the mean agent path length in Maze and the number of jumps in Mario, are difficult to predict. Interestingly, we observe that accurately predicting large number of jumps and long agent path length is harder compared to predicting them when the true value is low. Since the agent would be revisiting the tiles multiple times when the path length or the number of jumps is high, it becomes harder to obtain useful information from the occupancy grid.

We also believe that in these regions, minor environment differences could cause a large change in the measure value, making the prediction problem extremely difficult. For example, if a jump in Mario is

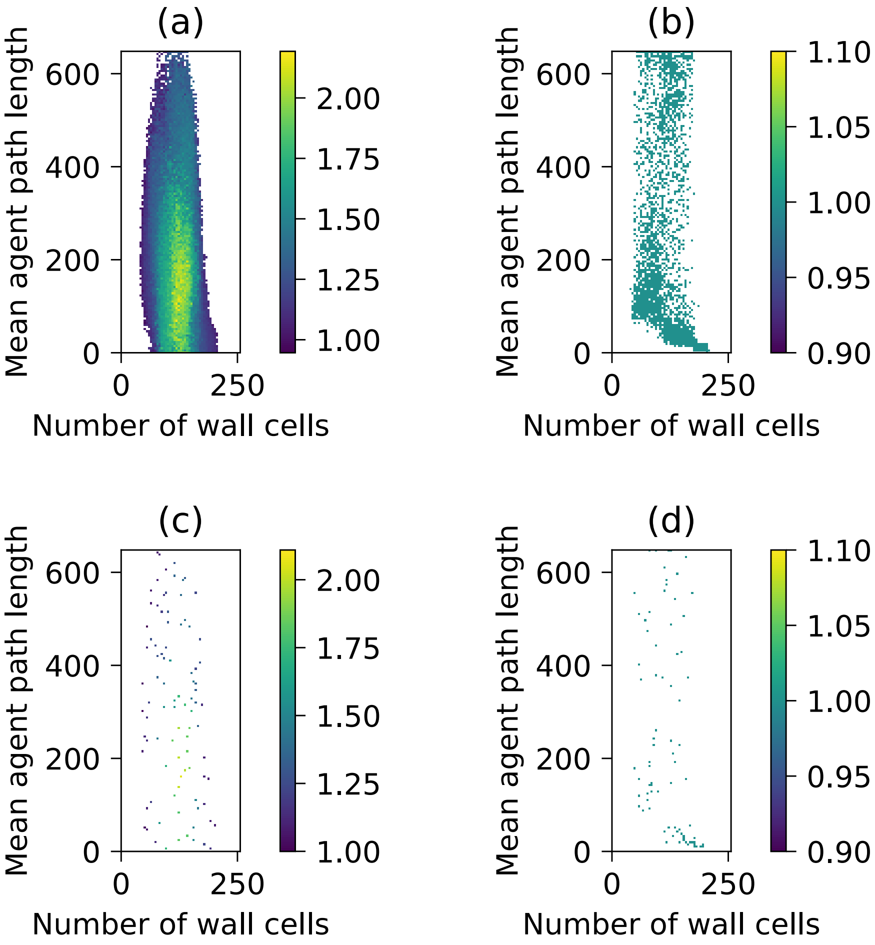

Figure 8: After running one surrogate model exploitation inner loop, we visualize the (a) surrogate archive, (b) positions of solutions from the surrogate archive in the ground-truth archive, (c) downsampled surrogate archive, and (d) positions of solutions from the downsampled surrogate archive in the ground-truth archive.

barely possible, the agent might need to try multiple times. But if one block is removed to make the jump easier, the agent might be able to finish it in one try, drastically reducing the total number of jumps.

Surrogate archive accuracy. To understand how the surrogate model's accuracy affects the creation of the ground-truth archive, we run an additional surrogate model exploitation inner loop starting from a completed DSAGE run and obtain a surrogate archive. We evaluate all the solutions in the surrogate archive and add them to a separate archive based on the ground-truth objective and measures. Additionally, we downsample the surrogate archive and create a corresponding ground-truth archive from the selected solutions.

Fig. 8 shows the full (8.a) and the downsampled (8.c) surrogate archive and the corresponding ground-truth archives (8.b, 8.d) in the Maze domain. We observe that many of the solutions from the surrogate archive end up in the same cell in the ground-truth archive, creating holes in the ground-truth archive. Only 47% and 41% of the solutions from the surrogate archive ended up in unique cells in the Maze and the Mario domains respectively. On the other hand, when downsampling, the percentage of surrogate archive solutions filling unique cells in the ground-truth archive improved to 97% and

94% in the Maze and the Mario domains respectively. Hence, downsampling reduces the number of unnecessary ground-truth evaluations.

In the Maze domain, only 0.06% of the surrogate archive solutions ended up in the exact same cell of the ground-truth archive as predicted. 4.6% of the solutions were in the 8×6 (the area from which downsampled solutions are chosen) neighborhood of the predicted cell. The average Manhattan distance between the predicted cell and the true cell was 53.8. In the Mario domain, 2.0% of the solutions were exactly in the same cell, 23.3% in the 5×5 neighborhood, and the average Manhattan distance was 14.2.

Despite the low accuracy of the surrogate model in terms of predicting the exact cell of the archive that the solution belongs to, the predictions were in the nearby region of the archive as evidenced by the average Manhattan distance. Furthermore, we conjecture that the holes in the ground-truth archive from a single outer iteration (as seen in Fig. 8.b, 8.d) are filled by solutions from other outer iterations. Hence, the final ground-truth archive (Fig. 3, Fig. 4) is more densely filled, leading to a better archive coverage and a better QD-score.

C Experimental Details

QD Optimization Algorithm. In the Maze domain, we used the MAP-Elites algorithm to generate the wall and the empty tiles of a maze. The first 100 solutions were generated by setting each cell to be either a wall cell or an empty cell uniformly at random. Every subsequent solution was generated by first choosing a random solution in the archive and mutating 10 random cells to a random value. The batch size was set to 150, i.e., 150 solutions were generated and evaluated in each iteration of the MAP-Elites algorithm. The archive was divided into 256×162 cells corresponding to the number of wall cells and the mean agent path length respectively.

In the Mario domain, we followed previous work [16] and selected the CMA-ME algorithm for QD optimization. The archive was divided into 150×100 cells corresponding to the number of sky tiles and the number of jumps respectively. The solutions, which are the input to a pre-trained GAN from previous work [16], were generated by 5 improvement emitters, each with a batch size of 30 and mutation power of 0.2.

In the baselines without a surrogate model, we ran the QD optimization algorithm until the number of ground-truth evaluations reached the given budget. For the other algorithms, we used the QD optimizer in the surrogate model exploitation phase and ran 10,000 iterations of the corresponding algorithm to create the surrogate archive.

We implemented all QD algorithms in Python with the pyribs [99] library.

Ancillary data and downsampling. In both domains, we recorded and stored the average number of visits by the agent to each discretized tile in the environment as the ancillary data. Algorithms using downsampling chose a single random elite from every 8×6 cells in the Maze domain and every 5×5 cells in the Mario domain

Surrogate Model Training. At the start of each outer iteration, the deep surrogate model was trained on the most recent 20,000 data samples for 200 epochs with a batch size of 64. The surrogate model was updated by backpropagating the mean square error loss between the predicted and the true objective, measures, and ancillary data. The model weights were then updated by the Adam [100] optimizer with a learning rate of 0.001 and betas equal to 0.9 and 0.999 respectively. We implemented the surrogate model with the PyTorch [101] library.

Computational Resources. For each algorithm-domain pair, we repeated the experiments 5 times and compared the mean performance. Experiments were run on two local machines and a high-performance cluster. The local machines had AMD Ryzen Threadripper with a 64-core (128 threads) CPU and an NVIDIA GeForce RTX 3090/RTX A6000 GPU. 16 CPU cores and one V100 GPU were allocated for each run on the cluster. Maze experiments without downsampling lasted for 4-5 hours while those with downsampling lasted for around 30 hours. Mario experiments without downsampling took 2-3 hours while those with downsampling took around 12 hours.

Table 3: QD-score and archive coverage attained by DSAGE variants with original hyperparameters and longer training versions of DSAGE Basic and DSAGE-Only Anc in the Maze and Mario domains over 5 trials.

	Maze		Mario	
Algorithm	QD-score	Archive Coverage	QD-score	Archive Coverage
DSAGE	16,446.60 ± 42.27	0.40 ± 0.00	4,362.29 ± 72.54	0.30 ± 0.00
DSAGE-Only Anc (longer training)	14,936.40 ± 400.45	0.36 ± 0.01	1,679.55 ± 213.21	0.11 ± 0.01
DSAGE-Only Anc	14,568.00 ± 434.56	0.35 ± 0.01	2,045.28 ± 201.64	0.16 ± 0.01
DSAGE-Only Down	14,205.20 ± 40.86	0.34 ± 0.00	4,067.42 ± 102.06	0.30 ± 0.01
DSAGE Basic (longer training)	12,618.20 ± 58.94	0.30 ± 0.00	1,983.84 ± 434.15	0.13 ± 0.03
DSAGE Basic	11,740.00 ± 84.13	0.28 ± 0.00	1,306.11 ± 50.90	0.11 ± 0.01

A single ground-truth evaluation in the Maze domain took between 1 to 13 seconds, with a mean of 3.5 seconds. The variation was mostly due to the difference in the agent performance since mazes that were finished in fewer steps required fewer forward passes through the agent's policy network. Evaluations in the Mario domain took between 1 to 135 seconds, with an average of 53 seconds, depending on the generated level. In contrast, a complete inner loop involving the surrogate model exploitation phase (around 1,500,000 surrogate evaluations) finished in around 90 seconds.

D Ablation: Effect of More Outer Iterations

We perform an ablation to test between two possible explanations for why having more outer iterations helps with performance: One explanation is that the larger number of training epochs, resulting from training the model in each outer iteration, itself helps with the accuracy of the surrogate model [102]. The second explanation is based on the fact that at the beginning of training, the surrogate model is inaccurate, and hence, the data generated by evaluating solutions in the surrogate archive would have been incorrectly predicted by the surrogate model. A larger number of outer iterations results in a larger number of times the algorithm updates the dataset with these adversarial examples, allowing the surrogate model to iteratively correct its own errors.

To disambiguate the two explanations, we increased the number of training epochs for the algorithms that do not use downsampling (DSAGE-Only Anc and DSAGE Basic), making the total number of training epochs the same as that with downsampling. In the Maze domain, the surrogate models of DSAGE-Only Anc and DSAGE Basic were trained for 5300 and 6400 epochs respectively in each outer iteration, compared to 200 epochs with downsampling. In the Mario domain, the models of DSAGE-Only Anc and DSAGE Basic were trained for 1350 epochs in each outer iteration, compared to 200 epochs with downsampling.

Table 3 shows the results with the longer training versions of DSAGE Basic and DSAGE-Only Anc. Longer training improves the QD-score and the archive coverage for both DSAGE Basic and DSAGE-Only Anc in the Maze domain and for DSAGE Basic in the Mario domain, but they still perform much worse than their counterparts with downsampling, DSAGE-Only Down and DSAGE. Hence, more iterative corrections of the errors of the surrogate model in variants with downsampling (due to a larger number of outer iterations) seems to be the major cause of performance improvement.

E Ablation: Random Selection of Surrogate Archive Solutions

As discussed in Sec. 6.3, selecting solutions from the surrogate archive with downsampling has several advantages which lead to better performance, with the major advantage being that downsampling increases the number of outer loop iterations. However, we could also increase the number of outer iterations by choosing a different subset selection mechanism, including simply selecting solutions uniformly at random. Thus, we test DSAGE with the random selection mechanism as an additional baseline. Namely, after every inner loop, we select a fixed number of solutions from the surrogate archive uniformly at random such that the number of outer iterations is approximately the same for both downsampling and random sampling.

Table 4 shows the results obtained by DSAGE with downsampling and random sampling. We observe that the performance with random sampling is lower than that of downsampling in the Maze domain, but they are very close in the Mario domain. Hence, we can conclude that increasing the number of

Table 4: Mean and standard error of the QD-score and archive coverage attained by DSAGE and DSAGE with random sampling in the Maze and Mario environments over 5 trials.

	Maze		Mario	
Algorithm	QD-score	Archive Coverage	QD-score	Archive Coverage
DSAGE	16,446.60 ± 42.27	0.40 ± 0.00	4,362.29 ± 72.54	0.30 ± 0.00
DSAGE (random sampling)	15,974.40 ± 78.71	0.39 ± 0.00	4,370.28 ± 107.87	0.30 ± 0.01

outer iterations is the largest contributor to the performance improvement, although downsampling has additional advantages that improve its performance in the Maze domain.

F Qualitative Analysis of the Algorithms

Fig. 9 and Fig. 10 show typical archives output by the algorithms in our experiments in the Maze and Mario domains, respectively.

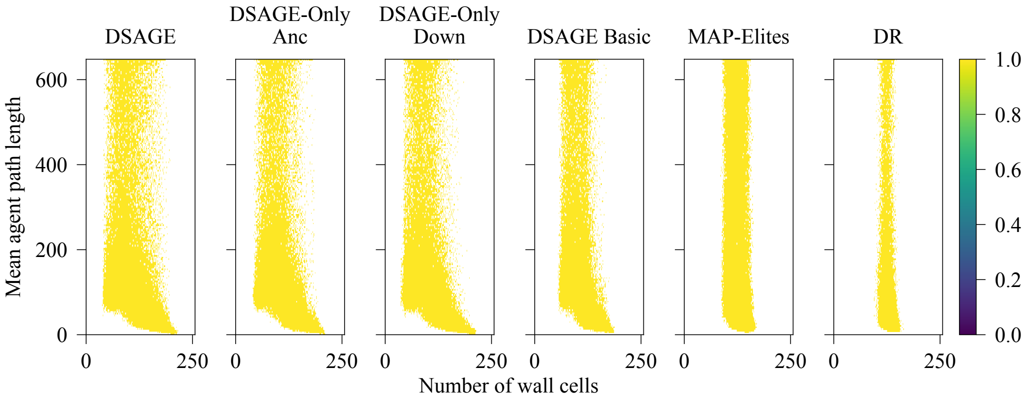

Figure 9: Example archives generated by algorithms in the Maze domain. Among the algorithms, DSAGE fills the largest portion of the archive, resulting in the highest QD-score, while MAP-Elites fills the smallest portion, resulting in the lowest QD-score. Since the objective only tests whether the level is valid, all levels in each archive have an objective of 1. Note that certain portions of the archive are physically impossible to obtain. For example, a maze with 0 wall cells would have the starting position and the goal at opposite corners, meaning that the mean agent path length must be at least 32.

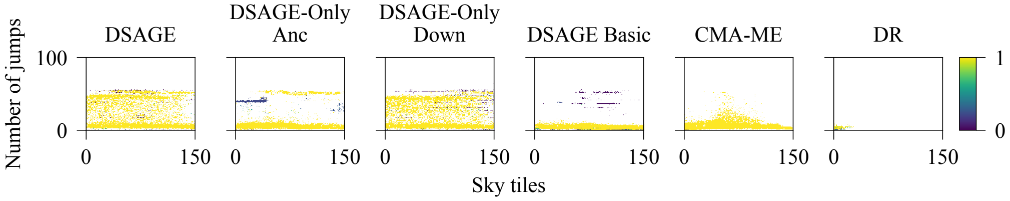

Figure 10: Example archives generated by all algorithms in the Mario domain. Compared to CMA-ME, DSAGE and its variants are more adept at finding levels with high numbers of jumps. The algorithms then differ in the objective values of the levels that have high numbers of jumps: DSAGE Basic finds levels with low objective values, so its QD-score is low. DSAGE-Only Down finds many levels with high numbers of jumps, but many of these levels have low objective values (hence the dark region in the top right of its archive), leading to a lower QD-score than DSAGE, which primarily finds levels with high objective values. Note that in our experiments, we never observed a level that caused Mario to jump more than 60 times, so the upper portion of all archives is unoccupied.

G Searching for Additional Agent Behaviors

Here we present example results from different measures in the Maze and Mario domains. By searching for these measures with DSAGE, we discover environments that elicit a wide range of agent behaviors not presented in our main paper.

G.1 Maze

Fig. 11, 12, and 13 show results from DSAGE runs in the Maze domain with different measures.

G.2 Mario

Fig. 14 and 15 show results from DSAGE runs in the Mario domain with different measures.

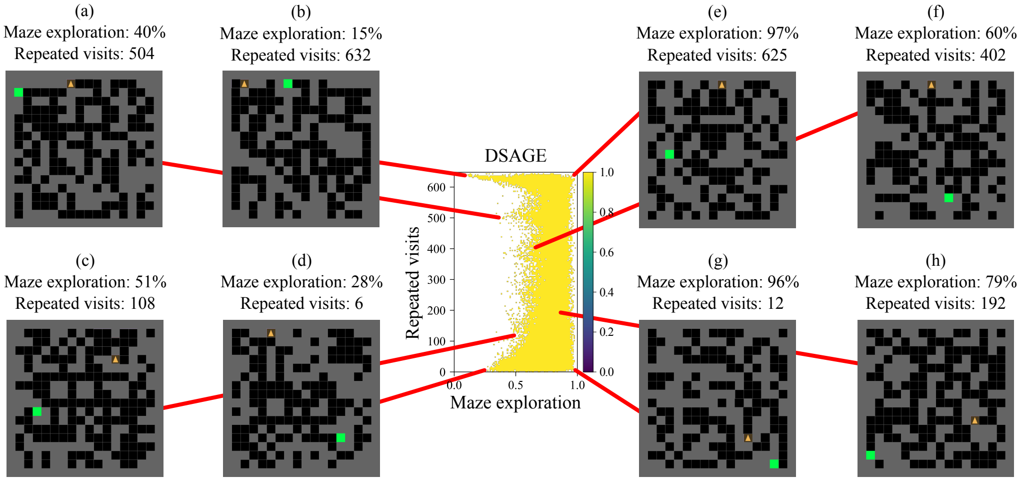

Figure 11: A DSAGE run in the Maze domain where the measures are (1) the fraction of reachable cells that the agent has visited (termed the “Maze exploration”; range [0, 1]) and (2) the number of times that the agent visits a cell it has already visited (termed the “Repeated visits”; range [0, 648]). Note that both of these measures are agent-based. In (a) and (b), the agent becomes stuck in a small portion of the maze, leading to many repeated visits but low maze exploration. Notably, the agent observes the goal multiple times in (b), but it never figures out how to go around the large wall which blocks it. In (c), the agent gets stuck in several traps (leading to repeated visits) but eventually makes its way to the goal. In (d), the agent heads directly to the goal, so it does not explore the maze much, and the only repeated visits it makes come from turning (when the agent turns, it stays in the same cell, which counts as a repeated visit). In (e) and (f), the agent visits multiple parts of the maze several times and is unable to reach the goal. In (g), the agent explores all of the space without revisiting many cells and eventually finds the goal. Finally, in (h), the agent has many repeated visits because it gets stuck at the beginning, but afterwards, it explores the rest of the maze and finds the goal. Refer to the supplemental material for videos of these agents.

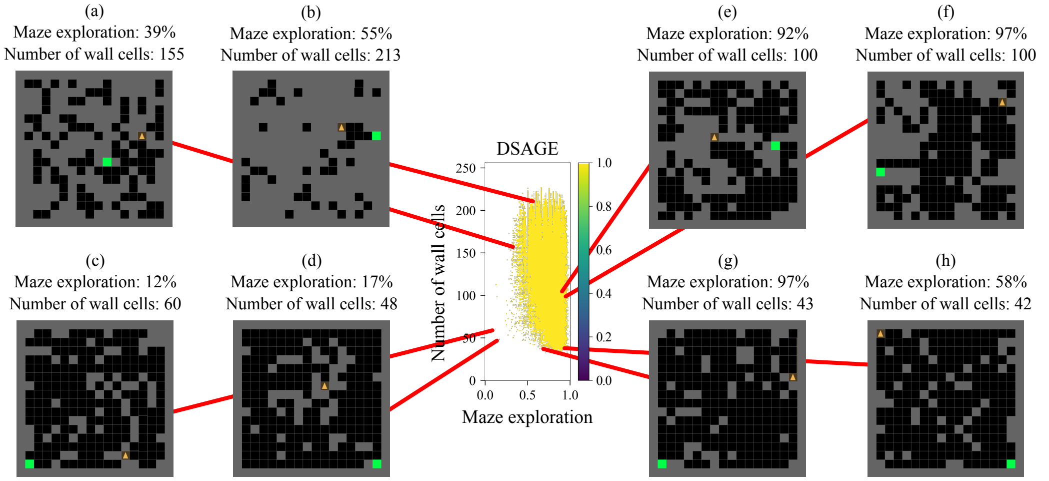

Figure 12: A DSAGE run in the Maze domain where the measures are (1) the fraction of reachable cells that the agent has visited (termed the "Maze exploration"; range [0, 1]) and (2) the number of wall cells in the maze (range [0, 256]). (a) and (b) are mazes where the wall cells define a straightforward path for the agent to follow to the goal, resulting in low maze exploration. In (c), the agent goes in circles in the bottom right corner, resulting in low exploration. (d) has a similar number of wall cells to (c), but the agent here is able to quickly find the goal, which also results in low exploration. (e) and (f) are two levels that are similar in terms of both measures yet have very different structures – in particular, (f) has a much larger reachable space for the agent to explore. In (g), the agent spends all its time exploring even though there are relatively few wall cells blocking its path. Finally, (h) has a similar number of wall cells as (g), but the agent heads almost directly to the goal. Refer to the supplemental material for videos of these agents.

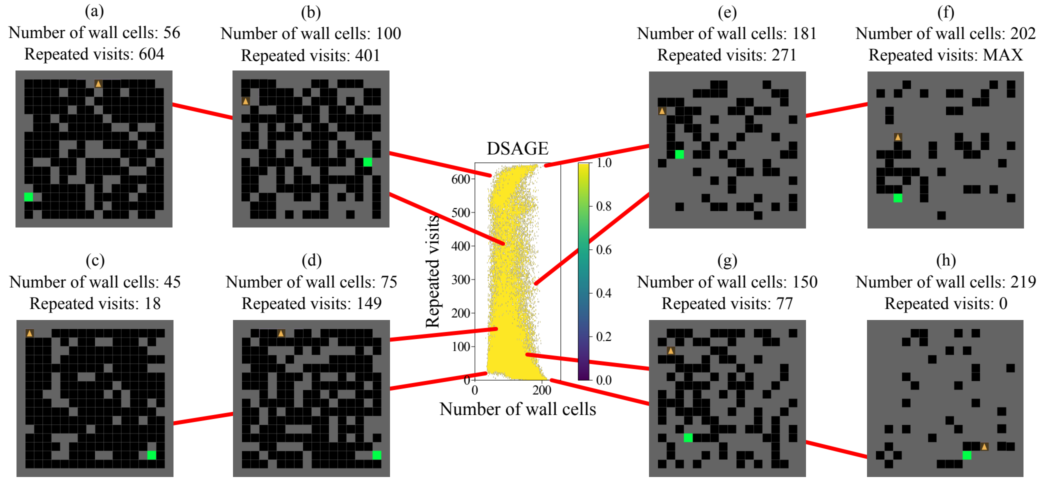

Figure 13: A DSAGE run in the Maze domain where the measures are (1) the number of wall cells in the maze (range [0, 256]) and (2) the number of times that the agent visits a cell it has already visited (termed the "Repeated visits"; range [0, 648]). In (a), the agent gets stuck in the top right corner since it is surrounded by walls with only one path out, so it has many repeated visits. In (b), the agent repeatedly goes around the maze and even sees the goal several times, but it usually does not reach the goal. (c) and (d) are relatively easy for the agent — since it finds the path quickly, it does not repeat many visits. (e) and (f) are cases where the agent gets stuck going in loops even though it is right next to the goal, which leads to many repeated visits. In (g), the agent makes several loops but eventually finds the goal. Finally, in (h), the agent goes directly to the goal, so it never repeats any visits. Refer to the supplemental material for videos of these agents.

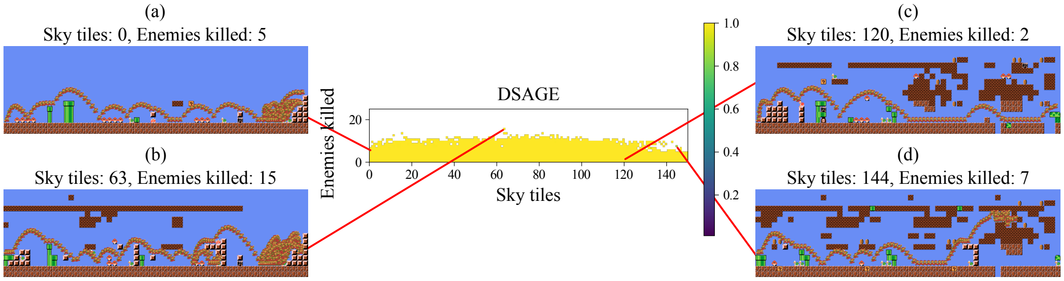

Figure 14: A DSAGE run in the Mario domain where the measures are (1) the number of sky tiles (exactly as in the main paper) and (2) the number of enemies Mario kills (range [0, 25]). Note that in (b), Mario kills many enemies because Mario repeatedly jumps on the bullets fired by the cannon at the end of the level. In (d), even though Mario kills multiple enemies, Mario cannot complete the level because the sky tiles form an unbreakable barrier. Refer to the supplemental material for videos of these agents.

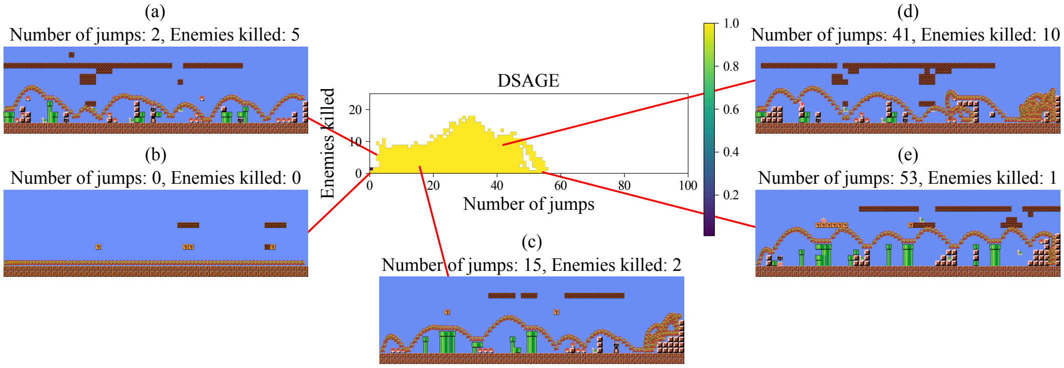

Figure 15: A DSAGE run in the Mario domain where the measures are (1) the number of times that Mario jumps (range [0, 100]) and (2) the number of enemies that Mario kills (range [0, 25]). Similar to the levels from our earlier experiment (Fig. 4(c)), levels (c), (d), and (e) here have a "staircase trap" at the end which causes Mario to perform many jumps, where different trap structures result in different numbers of jumps. Note that in some environments, there appear to be more jumps than indicated in the measures because Mario bounces whenever Mario lands on and kills an enemy, but these bounces do not count as jumps. Refer to the supplemental material for videos of these agents.