Supplementary Material: Structured agents for physical construction

A. Tasks details

A.1. Observation formats

For each task, the agent could use either an image-based observation, an object-based observation, or a combination of both as a segmentation-masks-based observation.

Object state observations are a list of vectors (one for each block), where each vector of size 15 contains information about the corresponding block position (x, y), orientation $(\cos(\theta), \sin(\theta))$, size (width, height), linear (v_x, v_y) and angular (v_θ) velocities, whether it is sticky or not, and one-hot information about its type (available block, placed block, target or obstacle). The list is ordered by the order under which objects appeared in the scene, but this information is discarded for the graph based agents. Information about which objects are in contact is also provided and is used when constructing the input for the graph based networks (see Sec. C).

Image observations start as 128×128 RGB renders of the scenes and are re-scaled down to 64×64 by averaging 2×2 patches, with the color channels normalized to $[0, 1]$. The x and y coordinate is also supplied for each point in the image and is normalized in the $[-1, 1]$ interval. The re-scaling procedure helps preserve spatial information at a sub-pixel level as color fading at the boundaries between the objects and the background.

Segmented images observations are a list of images, one for each block. They are obtained using a segmentation of the 128×128 render that maps each pixel to zero or more blocks that may be present at that pixel. Using this segmentation, we build a 128×128 binary mask for each block, re-scale it down to 64×64 by averaging 2×2 patches, and multiply it with the unsegmented RGB render to obtain per-block renders. We also add the mask as an additional alpha channel to the masked the RGB image, as well as coordinate channels.

A.2. Full scene generation and reward specifications

The full rendered scene spans a region a size of 16×16 (meters, unless otherwise indicated).

At the beginning of the episode, the agent has access to 7 available blocks: three small, three medium and one large block (corresponding to respective widths of 0.7, 2.1 and 3.5, all with height 0.7).

The physics simulation is run for 20 seconds after the agent places each block to make sure that the scene is at an equilibrium position before the score is evaluated, and before the agent can place the next block.

Silhouette: Each scene is comprised of 1 to 8 targets and 0 to 6 obstacles, arranged in up to 6 layers, with a curriculum over the maximum number of targets, maximum number of obstacles, and number of layers (see Fig. H.1). Levels are generated by (1) tessellating the scene into layers of blocks of the same sizes as the available blocks, with a small separation of 0.35, (2) sequentially (up to the required number of targets) finding the set of target candidates and sampling targets from this set (blocks in the tessellation that are directly on top of the floor or an existing target block) (3) sampling obstacles using a similar procedure. Both obstacles and targets that are further from the floor are sampled with higher probability to favor the generation of harder-to-construct towers and inverted pyramids. The average number of targets is 4.5 on the training distribution, and the number of targets goes up to 8 for the hardest levels. These numbers set an upper bound on the total reward that can be obtained. However, the average reward for an optimal agent is lower than that due to the cost of glue (silhouettes generated using this procedure are not guaranteed to be stable, thus the best possible solution may require glue). We ran a baseline (using the action interface of the Relative GN-DQN agent) consisting of two heuristics for deciding (a) where to place blocks: at target positions, sequentially layer by layer, and from center to the sides and (b) when to use sticky blocks: whenever none of the existing blocks that would touch the new block are sticky and the center of mass of the new block would not be supported by the blocks in the previous layer. This baseline achieves a reward of 3.42 on the training distribution and 5.27 on the hardest levels. For comparison our best GN-DQN-MCTS agent seed on this task achieves 4.15 and 7.18, respectively.

Connecting: There are at most three vertical layers of obstacles above the floor and a layer of three targets above the highest obstacles. Each layer consists of up to three obstacles, whose lengths are uniformly and independently sampled from the interval $[0.7, 2.8]$. The layers of obstacles separated by enough distance for one block can be placed between any two layers of obstacles. The curriculum is comprised of scenes fewer obstacle layers, while the number of targets is unchanged (see Fig. H.4 for examples). Since glue is unpenalized, the maximum reward available to the agent is exactly 3. We expect a heuristic based on path finding would achieve the total reward of 3.

Covering: There are at most three vertical layers of obstacles above the floor at any location, and up to 2 obstacles in each layer, with lengths uniformly and independently sampled from the interval $[0.7, 2.8]$. As in *Connecting*, these layers are well separated so that one block can be placed between any two layers of obstacles. The curriculum is comprised of scenes with obstacles only in the first two lower layers (see Fig. H.2). The total available length to cover is 5.25 on the training distribution and 7.88 for the hardest levels. This provides a tight upper-bound on the maximal reward and the agent could be expected to achieve this. We ran a baseline (using the action interface of the Relative GN-DQN agent) consisting of placing objects layer by layer, prioritizing large blocks and using the following heuristics for (a) odd layers (those with obstacles): place as many blocks as possible in gaps between obstacles, (b) even layers: placing objects to cover the obstacles from the previous layer, then to fill the remaining gaps when possible. This baseline achieves a reward of 3.85 on the training distribution and 5.31 on the hardest levels. For comparison our best GN-DQN-MCTS agent seed on this task achieves 4.65 and 7.18 respectively.

Covering Hard: There are at most two vertical layers of obstacles above the floor at any location, and up to 2 obstacles in each layer, with lengths uniformly and independently sampled in $[0.7, 3.5]$. The curriculum is comprised of scenes with only one layer of obstacles (see Fig. H.3 for examples). The layers of obstacles are closer to each other than in they were in *Connecting* or *Covering Hard*. The maximum length that can be covered is 4.2 on the training distribution and 6.3 on the hardest levels, but this only gives a weak upper bound on the possible reward because of the cost of glue and limited supply of blocks. Given the additional complexity of this task, involving resource planning (which blocks to save for later), and balancing the use of sticky blocks (pay price) with the use of arches (use more blocks), we did not find a simple heuristic that could provide a relevant baseline. In particular, note that our heuristic for *Covering* is not relevant here as it assumes an infinite supply of blocks.

Curriculum complexity: Curricula were designed to increase in complexity while preserving instances of scenes from previous points in training to avoid catastrophic forgetting. This allows us to make a distinction, for any task and curriculum level, between *Hardest Scenes* (scenes types that are introduced for the first time at the present level) and *All Scenes* (training distribution, including hardest scenes at the current level and lower level scenes). Additional details about the conditions for advancing through the curricula are given in Sec. D for the DQN agents and Sec. F for the RS0 agents.

B. Implementation details of the action specification.

Continuous absolute actions are 4-tuples (X, x, y, s). X is compared to the x-coordinates of each of the available blocks and the closest block c is chosen. A new block identical to it is then spawned with its center at location (x, y). The resulting object is sticky if and only if the continuous action $s \in [-1, 1]$ is positive.

Discrete absolute actions are 4-tuples (u, i, j, s). $u \in \{1, 2, \ldots, 7\}$ is an index within the set of available blocks to decide which block will be placed on the scene. i, j are discrete index to place this block on the scene. Specifically the scene was discretized in height$\times$width using different sizes from 8×64 to 256×256, finding the best results for 256×256 in *Silhouette* and 8×64 for the other tasks. s is a discrete variable in $\{-1, 1\}$ indicating whether the placed object should be made sticky or not.

Continuous relative actions are 5-tuples, $(X, x, y, \Delta x, s)$. Here X, s have identical meaning to the absolute case, and c is again the object selected by X. The object r whose center is closest to (x, y) is then selected as reference. Then, the x-coordinate of the placed block x_p is determined by $\Delta x \in [-1, 1]$, such that $x_p = x_r + \Delta x \left(\frac{w_r + w_c}{2} + \epsilon_x\right)$, where x_r is the x-coordinate of the center of r, w_r and w_c are the widths of the objects r and c, and ϵ_x is a small offset so that the objects are not touching laterally.

If r is a target object centered at (x_r, y_r), the y-coordinate of the center of c will be placed at $y_p = y_r + \epsilon_y$ so that c is vertically overlapping with r (where ϵ_y is a small offset so that the objects are not perfectly flush). If r is a solid object, c is placed just above r, i.e. $y_p = y_r + \frac{h_r + h_c}{2} + \epsilon_y$, where h_c and h_r are the heights of the objects c and r, respectively. If the agent chooses an invalid edge (where c is not an available block, or where r is not a block in the scene), then the episode is

terminated with a reward of zero. We use $\epsilon_x = \epsilon_y = 0.04$ throughout.

Discrete relative actions are triplets, (e, i, s), where $e := (c, r)$ is an edge in the structured observation graph between the chosen new block and the selected reference block, i is an index over fine discretization of discrete horizontal offsets to place the chosen block relatively to the reference block, and s is as before. If the blocks c are not an available block or that the block r is not a block already in the scene, then the episode is terminated with a reward of 0.

For the x offsets, we use a uniform grid with n bins in the range $[-\frac{w_r+w_c}{2}(1+\frac{1}{n-3}), \frac{w_r+w_c}{2}(1+\frac{1}{n-3})]$, where w_r and w_c are as before (this is such that there are exactly n segments in the range $[-\frac{w_r+w_c}{2}, \frac{w_r+w_c}{2}]$). We then pick the i-th value in this grid as the relative x position. The y coordinate of the placed block is computed as before, but we also experimented with predicting the relative offset Δy, and varying the number of discrete offsets n (see Sec. D for details).

C. Architecture details

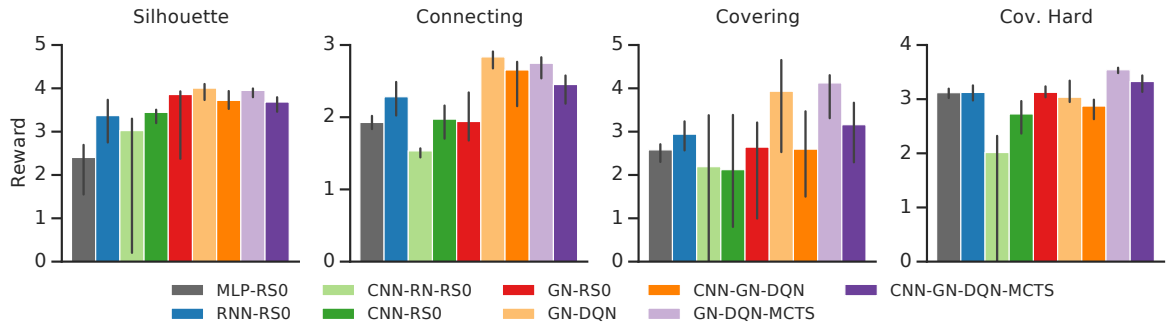

Figure C.1. Overall comparison of all agents tested, including MLP and relation network (Santoro et al., 2017) baselines, averaging across curriculum levels. Bars show median performance across seeds, and errorbars are min and max seeds.

C.1. MLP-based architectures

MLP: The pre-processor of the MLP model consists of concatenating the list of blocks as given by the environment (blocks, available blocks, obstacles, targets) padded with zero blocks (up to the total maximum number of objects in each task with a 1 hot indicator of padding), and normalizing it with a LayerNorm layer into a fixed set 100 features. This fixed size vector is then processed by the core MLP consisting of four hidden layers of 256 units with ReLU non-linearity, and an output layer to match the required output size. We found this MLP model to have equal or worse performance to the RNN agent, and thus did not report results on it in the main text; however, Fig. C.1 includes results for the MLP agent across the tasks.

RNN: The RNN model pre-processor uses a GRU (hidden size of 256) to sequentially process the objects in the the scene (including padding objects up to a maximum size as described in the MLP). The output of the GRU after processing the last object is then used as input for the core MLP (identical in size to the on described in the MLP model). In some generalization settings, where the total number of objects increased drastically, we found better generalization performance by clipping/ignoring some of the objects in the input, than by allowing the network to process a longer sequence of objects than used at training time.

CNN: The CNN model pre-processor passes the 64×64 input image through a 4-layer convolution network (output channels=[16, 32, 32, 32]) followed by a ReLU activation, a linear layer on the flattened outputs into embedding size of 256, and another ReLU activation. Each layer is comprised of a 2d convolution layer (size=3, stride=1, padding="same") and a max pooling layer (size=3, stride=2, padding="same"). The vector embedding of the image is then processed by and MLP core (identical in size to the on described in the MLP model, except that it uses 3 layers instead of 4).

CNN-RN: We found this CNN-RN model to have equal or worse performance to the vanilla CNN agent, and thus did not report results on it in the main text; however, Fig. C.1 includes results for the CNN-RN agent across the tasks. We use a higher-resolution convolutional feature map, using residual connections to increase depth and ease training. Each residual block with N channels consists of a N-channel (size=3, stride=1, padding="same") convolution and a max pool (size=3, stride=2, padding="same"). This is followed by a N-channel convolution (size=3, stride=1, padding="same"), a ReLU, and

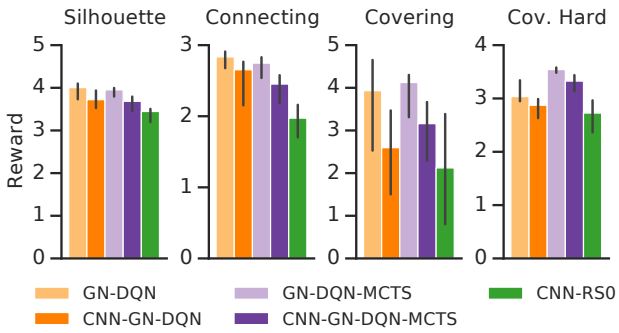

Figure C.2. Comparison of object- and pixel-based agents, averaged across curriculum levels. Bars indicate median performance across seeds, and errorbars are min and max seeds. The CNN-GN-DQN agent augmented with a vision module is able to perform almost as well as its object-based counterparts, and clearly above the CNN-RS0 agent. The same holds for the CNN-GN-DQN agent trained with MCTS, which sometimes even outperforms the object based, model free DQN.

another N-channel convolution (size=3, stride=2, padding="same"), the output of which is added to the max pool output to get the block output. We apply 3 such blocks with N=[16,32,8]. This gives us a vector of length 8 at every spatial location, to which we apply the standard Relation Net architecture (Santoro et al., 2017): we contatenate each pairs of these vectors, and feed the length-16 vector into a 2-layer MLP with ReLU activations (64 and 128 units), before applying an average pool operation over all pair representations. This 128-length vector is a linear layer to produce the final embedding of size 256.

C.2. GN-based architectures

Graph pre-processing: We use the list of objects or segmentation masks to construct the graphs that are input to the RS0-GN and DQN-GN agents, only discarding the information about the order of appearance of the object in the scene.

For the RS0 agent, we then construct a sparse graph from this set of nodes by connecting (1) available objects to all other objects in the scene; (2) targets and obstacles to all blocks in the scenes; and (3) blocks that are in contact. The DQN agent takes a fully-connected graph as input but we also experimented with feeding it the sparse representation (see Sec. D.3 for details).

GN architecture: We use the encode-process-decode architecture described by Battaglia et al. (2018). comprised of an independent graph encoder, a recurrent graph core with separate MLPs as node, edge, and global functions followed by three GRUs, respectively, and finally as a decoder either a graph network (for the RS0 agent) or graph independent (for the DQN agent). In symbols, given a graph observation o, we process it as

$$\begin{aligned} e &= E(o) & o_0 &= e \qquad o'_0 = e \\ o_n &= G([o, o_{n-1}, o'_{n-1}]) & o'_n &= R(o_{n-1}) \\ d &= D(o'_n) & & (1 \le n \le n_{\text{rec}}) \end{aligned}$$

where E and D are independent graph network (see Battaglia et al. (2018)), G is a full graph network, and R is a recurrent independent graph network. We use two hidden layers of 64 units with ReLU non-linearity within all our graph networks.

For this discrete agent, the Q values are finally decoded from d as

$$q = M([x, d_{\text{globals}}]_{x \in d_{\text{edges}}}),$$

similarly to the approach of Dai et al. (2017).

For the RS0 agent we find that having more than 1 recurrent steps in the recurrent graph core did not improve performance so we use a single recurrent step, and disabled the GRU (no longer needed without recurrent steps).

Segmented images pre-processing: In the case of the Segmented images observations, each of the nodes in the graph contains an image, which we process independently using a pre-processor similar to that of the CNN model, but smaller (three layers with [8, 16, 8] output channels, followed by two activated linear layers with sizes [64, 32]). This produces a graph with 32 embedded features for each node.

C.3. Comparison of pixel based methods with objects based methods

We compare pixel based approaches with object based approaches on Fig. C.2, emphasizing that the graphical networks that take segmented images as input fare closer to their object based graphical counterparts than to raw CNNs, making their usage an exciting avenue for future work.

We also implemented use a CNN followed by a relation network (Santoro et al., 2017), but it fared worse than the vanilla CNN in all experiments so we excluded it from the main text (see Sec.C.1 of the Supplemental).

D. Further study on the GN-DQN agent

D.1. General implementation

We implement a DQN agent with a structured graph input and graph output (roughly similar to Dai et al. (2017)), but where the Q-function is defined on the edges of the graph. This agent takes a fully-connected graph as input. The actions are decoded from the edges (resp. the global features) of the network's output in the case of the discrete relative (resp. absolute) agent. The learner pulls experience from a replay containing graphs, with a fixed replay ratio of 4. The curriculum over scene difficulty is performed on a fixed, short schedule of 4×10^4 learner steps. The main difference with respect to a vanilla DQN is the way we perform ϵ-exploration, which we explain in more detail below.

We use a distributed setup with up to 128 actors (for the largest MCTS budgets) and 1 learner. Our setup is synchronized to keep the replay ratio constant, i.e. the learner only replays each transition a maximum number of times, and conversely actors may wait for the learner to be done processing transitions. This results in an algorithm which has similar learning dynamics to a non-distributed one.

D.2. ϵ-exploration schedule

The majority of actions of the discrete agent are invalid, either because they (1) do not correspond to an edge starting from an available block and reaching to an already placed object; (2) because the resulting configuration would have overlapping objects; or (3) because the resulting scene would be unstable. This has the consequence that doing standard ϵ-exploration strongly reduces the length of an episode (longer episodes are exponentially suppressed), effectively performing more exploration at the beginning of an episode than at its end. To counteract this effect, we use an adaptive ϵ-schedule, where the probability of taking a random action at the n-th step of an episode is given by $p_n = \frac{\epsilon}{\min(\widehat{L}-n,1)}$, where $\widehat{L}$ is an empirical estimate of an episodes typical length, and we use $\epsilon = 0.3$ throughout the paper. The final performance is mostly unchanged, but we observe that this makes learning faster and helps with model training (see Sec. E.3).

D.3. Effect of the number of propagation steps and graph connectivity in the graph network

The results reported elsewhere in this text for the discrete agent were all obtained with $n_{\text{rec}} = 3$ (see Sec. C.2) and a fully-connected input graph, but we experimented with varying n_{rec} and changing the graph connectivity. In Fig. D.1 we show that performance improves with the number of recurrences, but that training is also more unstable, as demonstrated by the wider shaded area around the curve. Empirically, $n_{\text{rec}} = 3$ provides the better compromise between performance and stability.

Those results were all obtained with a fully-connected graph, with a number of edges therefore equal to the number of objects squared. Many of those edges do not however correspond to valid actions or to directly actionable connections, and we experimented with removing those edges from the graph, using the same sparse graph used by the RS0 agent and described in Sec. C (note that this graph typically has about 4 times fewer edges than the fully-connected one). What we observe is that this reduces the reasoning capacities of the discrete agent and therefore decreases performance. Augmenting the number of recurrences can partially correct this effect: the best seed with a sparse graph and $n_{\text{rec}} = 7$ can get to the same level of performance as a seed of the fully-connected graph with $n_{\text{rec}} = 3$), but this then happens at the detriment of training stability.

D.4. Relative actions on both the x and y axes

Our discrete relative agents must choose a block to place, an object to use as a reference, and an offset relative to that reference. Thus far, that offset is only in the x-direction, since a small y-offset above the reference block is almost always

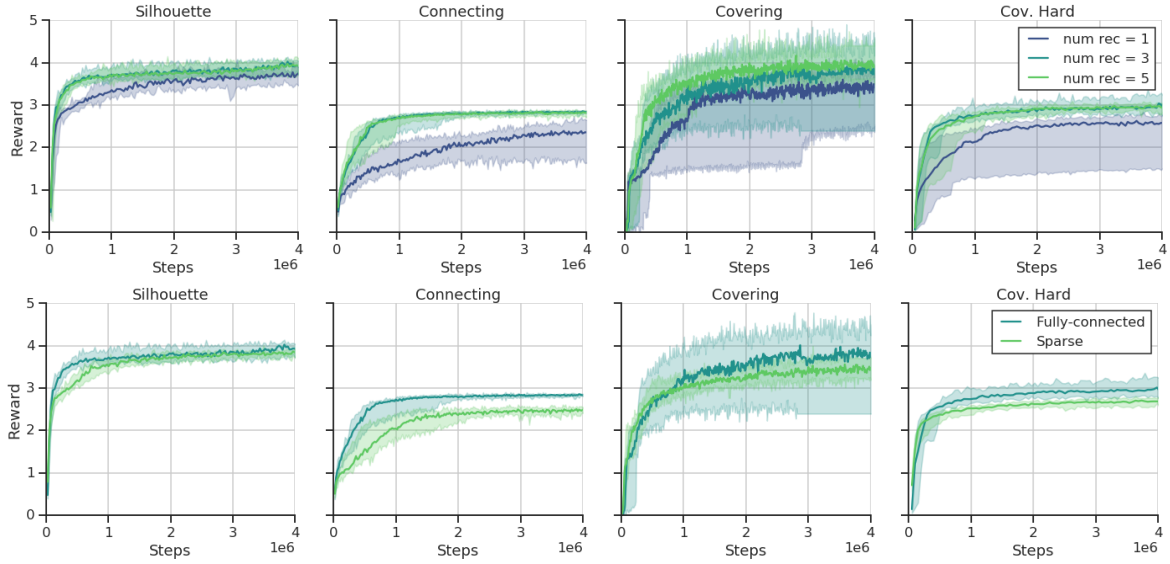

Figure D.1. **Top panel**: Influence of the n_{rec} parameter on the discrete agent's performance. **Bottom panel**: Influence of using a sparse graph or the fully-connected graph. Results are qualitatively similar when considering the agents trained with MCTS. In all plots, solid lines are median performance across seeds and shaded regions extend between min and max seeds.

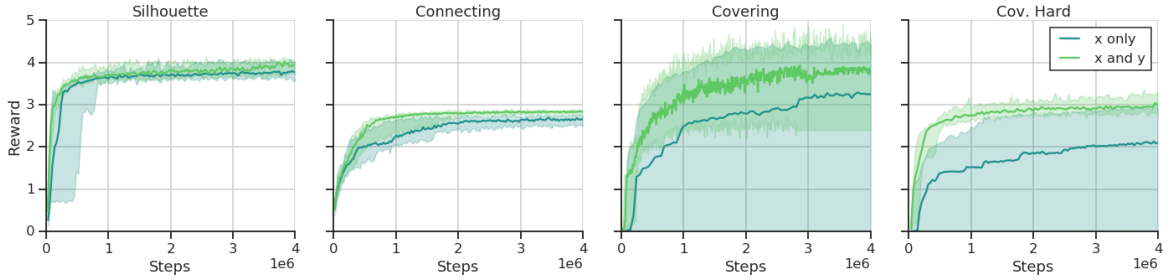

Figure D.2. Influence of predicting both the x and y relative coordinates or not. Observe that best performance reaches comparable values in both cases. In all plots, solid lines are median performance across seeds and shaded regions extend between min and max seeds.

sufficient. However, what happens if we allow the agent to choose y offset as well? We observe that this multiplies the size of the action space by the number of discretization points (in our case, 15), therefore making learning harder. On the other hand, for seeds that manage to start learning, the final performance is equivalent that of the agent which only predicts the relative x position (see Fig. D.2), despite a number of actions much larger than that of a typical discrete agent, as shown in Table D.1.

D.5. Number of discretization steps

The architecture of the GN-DQN agent naturally represents discrete quantities (i.e., choosing blocks out of a fix set), but using a discrete x-offset loses precision over outputting a continuous value. In order to probe the effect of this approximation, we varied the number of discrete locations that the agent is allowed to choose as the second dimension of the action (Fig. D.3). We observe that a finer discretization of the space allows for slightly better final performance on some problems, but also implies a slower and more unstable learning. Empirically, the 15 steps of discretization used in this paper offers the best compromise. An interesting avenue for further research would be to create an agent that can produce continuous actions attached to a particular edge or vertex of the input graph.

Other parameters: In all the paper, and unless otherwise specified, we fix the learning rate of the discrete agent to 10^{-4} (resp. $2\ 10^{-4}$) for the model-free (resp. model based) agent and use the Adam optimizer. We use a batch size of 16 and a

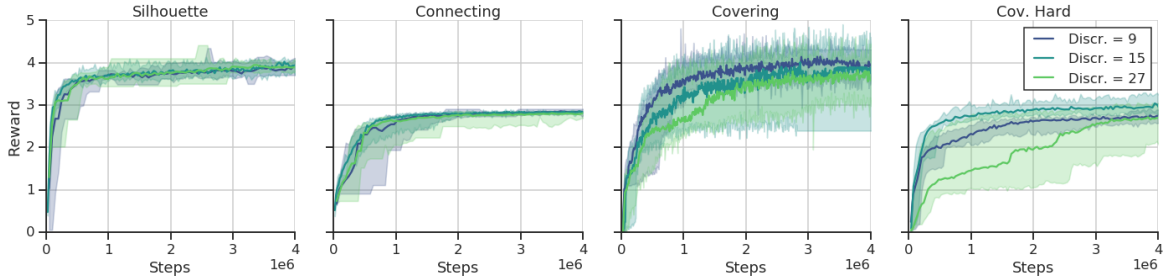

Figure D.3. Influence of the number of discretization steps for the relative horizontal positioning of the discrete agent. In all plots, solid lines are median performance across seeds and shaded regions extend between min and max seeds.

	Silhouette	Reaching	Covering	Covering Hard
absolute	$9\ 10^5$	$7\ 10^3$	$7\ 10^3$	$7\ 10^3$
relative	$2\ 10^4$	$2\ 10^4$	$2\ 10^4$	$3\ 10^3$
relative (x, y)	$2\ 10^5$	$3\ 10^5$	$3\ 10^5$	$5\ 10^4$

Table D.1. Typical number of actions for the variations of the tasks, for the discrete agents. The first line reports the number of actions for the best performing discrete absolute agent, the second line for the main discrete relative agent, and the third line for the agents predicting both x and y relative positions.

replay ratio of 4. We perform a linear curriculum over the problem difficulty over a short amount of steps (4×10^4 learner steps). We run all model free agents for 10^7 learner steps, i.e. approximately 2.5×10^6 actor steps. Model based agent are run for up to 4×10^6 learner steps (10^6 actor steps). Every experiment is run with 10 different seeds.

E. Further study on the GN-DQN-MCTS agent

E.1. Details of MCTS

The efficiency of Monte-Carlo Tree Search (MCTS) (Coulom, 2006) planning in RL has recently been highlighted in (Guo et al., 2014; Silver et al., 2016; 2017; 2018). Here we combine our DQN agent with MCTS, in the spirit of Sutton (1991) and Azizzadenesheli et al. (2018). We define a state s in the tree by the sequence of actions that led to it. In other words, given an episode starting with a configuration s_0 and a sequence of actions $(a_0, .., a_t)$, we simply define $s_t := (a_0, .., a_t)$ (we do not try to regroup states that would correspond to the same observation if coming from different actions sequences). Each node in the tree has a value estimated as

$$V(s) := \frac{1}{N(s)} \left(\max_a Q(s,a) + \sum_{r \in \text{rollouts}, s \in r} \hat{Q}(r,s) \right), \qquad N(s,a) = 1 + \sum_{r \in \text{rollouts}, s,a \in r} 1, \qquad N(s) = \sum_a N(s,a) \tag{1}$$

(observe that the resulting Monte-Carlo tree has a variable connectivity). In this expression, $\hat{Q}(r,s)$ is the standard Monte-Carlo return of a rollout after state s. The left term $\max_a Q(s,a)$ acts as a prior on the value of a node s. It is essential to include this term to obtain learning with MCTS, even if using a large budget (see Fig. E.1). We interpret this as being due to the large number of actions stemming from each node and to the fact that many of these actions are actually invalid.

We then perform MCTS exploration by picking the action a that maximizes $\mathcal{V}(s,a)$, where for common MCTS with UCT exploration (Kocsis & Szepesvári, 2006) one would have

$$\mathcal{V}(s,a) = V((a_0, \dots, a_t, a)) + c\sqrt{\frac{\ln N(s)}{N((a_0, \dots, a_t, a))}}$$

Remembering that the action a can be decomposed as (α, β), where α represents an edge index and β all the remaining

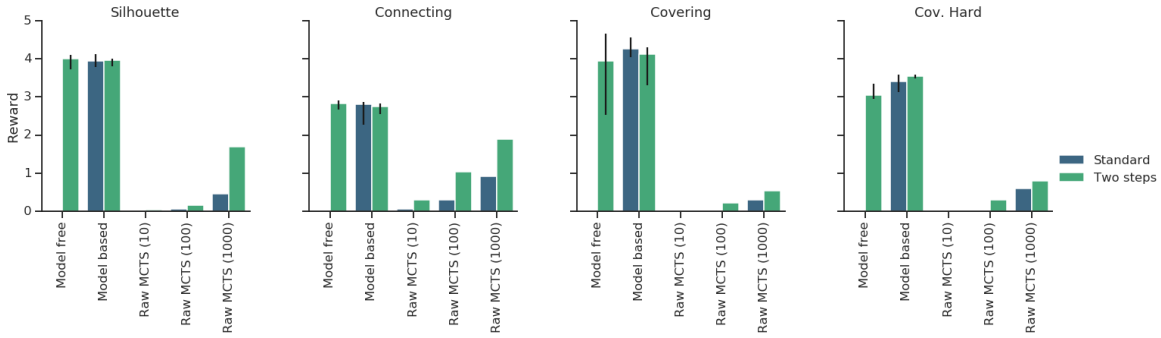

Figure E.1. Comparison of the learning based approaches with non-guided MCTS. Model free denotes the relative DQN agent, model based the relative DQN agent with MCTS at training time (budget of 10), Raw MCTS (10) (resp. Raw MCTS (100), Raw MCTS (1000)) denotes a pure MCTS search (without prior on an action value) with a budget of 10 (resp. 100, 1000). For each of the MCTS result, we show the result when performing the search over the actions in two stages (as described in E) or in the usual way. The non-visible bars correspond to zero reward.

dimensions of the action (relative x placement, use of glue or not, ..), we instead first pick α as the maximizer of

$$\mathcal{V}'(x,\alpha) := \max_{\beta} \left[V((a_0,\dots,a_t,(\alpha,\beta))) \right] + c\sqrt{\frac{\ln N(s)}{\sum_\beta N((a_0,\dots,a_t,(\alpha,\beta)))}},$$

and then β as the maximizer of $\beta \to \mathcal{V}(s,(\alpha,\beta))$. We find this approach to yield slightly better results (see Fig. E.1), and to offer better invariance to changes in the second dimension of the action (e.g. when introducing two dimensional relative placement or changing the number of discretization steps). We use a value of $c = 2$ for the UCT constant, and do not find a strong influence of this value on our results.

We then use a transition model to deduce the observation, reward and discount obtained when transitioning from s to s'. For the results presented in the main part of this work, we focused on using a perfect transition model, obtained from reseeding the environment every time with the initial state of an episode and reapplying the same sequence of actions. While this is impractical for the large MCTS budgets used in some other works, this provides an upper-bound on the performance that can be obtained with a learnt model and allows to separate hyper-parameters analysis. Also, as we will show in the next paragraph, it is possible to obtain significant gains even when performing the MCTS expansion only at test time.

E.2. Results with the environment simulator

We incorporate planning in two ways to our relative discrete agent. In the first variation, we only perform MCTS at test time, using an independently trained Q-network to act as a prior in our MCTS expansion (cf. Eq.(1)). We observe that this improves the results on almost all problems but for *Covering*. In particular, in *Reaching*, the fraction of the hardest scenes where the agent does not reach all three targets is decreased by a factor of 4 (from 55% down to 16%).

In the second variation, we also perform MCTS at training time: the actor generates trajectories using MCTS expansions using its current Q-function, and the resulting trajectories are then fed to the learner (which does not do any Monte-Carlo sampling). We observe that this second approach yield slightly more stable learning and higher performance on *Covering Hard*, the task that requires the more reasoning (see the last panel of Fig. 5). On the other hand, on other problems, it yields a similar or even decreased performance.

An interesting point to note is that, when training with a perfect simulator, the transfer into the Q-function is very imperfect, as demonstrated by the low value of the left most point on the darker curve of Fig. 5. As it turns out, the agent is relying on the model to select the best action out of the few candidates selected by the Q-function. This may explain why the performance does not necessarily increase when testing with more budget, as the Q-function does not in this case provide a good prior when doing a deeper exploration of the MCTS tree. This is, in essence, also similar to the hypothesis put forward in (Azizzadenesheli et al., 2018).

E.3. Results with a learned model

Finally, we extend the previous model-based results by performing the MCTS expansion with a learnt model rather than a perfect simulator of the environment. Using a learnt object-based model was recently put forward as an efficient method for planning in RL (Pascanu et al., 2017; Hamrick et al., 2017) and for predicting physical dynamics (Sanchez-Gonzalez et al., 2018; Janner et al., 2019). Note, however, that none of these approaches have attempted to use MCTS with a graph network-based model.

The model is an operator taking as an input a graph observation and an action, and outputting a new graph observation alongside a reward and discount for the transition:

$$M : \mathcal{O} \times \mathcal{A} \to \mathcal{O} \times \mathbb{R} \times [0, 1]$$

Given a sequence of observations o, actions a, rewards r and discounts γ belonging to a single episode, we train this model with an unrolled loss

$$L_{n_{\text{unroll}}}((o_{t'})_{t \leq t' \leq t+n_{\text{unroll}}}, (r_{t'}, a_{t'}, \gamma_{t'})_{t \leq t' \leq t+n_{\text{unroll}}-1}) := \sum_{n=0}^{n_{\text{unroll}}-1} l(M^{(n)}(o_t, (a_{t'})_{t \leq t' < t+n}), r_{t+n}, \gamma_{t+n})$$

where we defined the predicted observation after n steps

$$M^{(n)}(o_t, (a_{t'})_{t \leq t' < t+n}) = M(M^{(n-1)}(o_t, (a_{t'})_{t \leq t' < t+n-1}), a_{t+n-1})_0, \qquad M^{(0)}(o_t) = o_t$$

and the single step loss

$$l((o, r, \gamma), o', r', \gamma') := \|o - o'\|_2 + \|r - r'\|_2 + D_{\text{KL}}((\gamma, 1-\gamma)||(\gamma', 1-\gamma')).$$

In practice we varied the number of unrolls n between 1 and 4. The model training is slower with a larger number of unrolls, but it yields more consistent unrolls when used within the MCTS expansion (ideally, the number of unrolls should probably match the typical depth of a MCTS unroll). The model architecture is similar to the one of the main Q-network described in Sec. C.

Model pre-training: At first, we experiment with using a pretrained, learnt model to then perform Q-learning with MCTS. The setup is therefore as follows:

(1) Train an agent model free, or with a perfect environment simulator.

(2) Train a model on trajectories generated by this agent.

(3) Train a second agent with the model learnt in (2)

We observe in Fig. E.2 that this allows to obtain an improved performance at the beginning of training, matching the results obtained with a perfect environment simulator. However, on longer timescales, the performance plateaus and does slightly worse than a model free agent. We interpret this as being due to the rigidity of the model on longer timescales, which is not able to generalize enough to the data distribution that would be required to obtain larger rewards.

Model learnt online: Finally, we try to learn a model online. In this case the agent is trained with a model which is learnt at the same time on trajectories generated by the agent. As shown on Fig. E.3, we are able to slightly outperform the model free agent on short timescales in two of the problems (Silhouette and Covering Hard), while the noise introduced by the model is prohibitive again in Covering. On longer timescales, the imperfections of the model make the agent trained with a learnt model converge to the same rewards as the one trained without a model, rather than with a perfect model.

We believe that both in this case and when pre-training the model, understanding how to better train the model so that it generalizes better and yields sharper predictions are important areas of future research, and we see the positive results described here at the beginning of training as a strong motivation to pursue work in this direction.

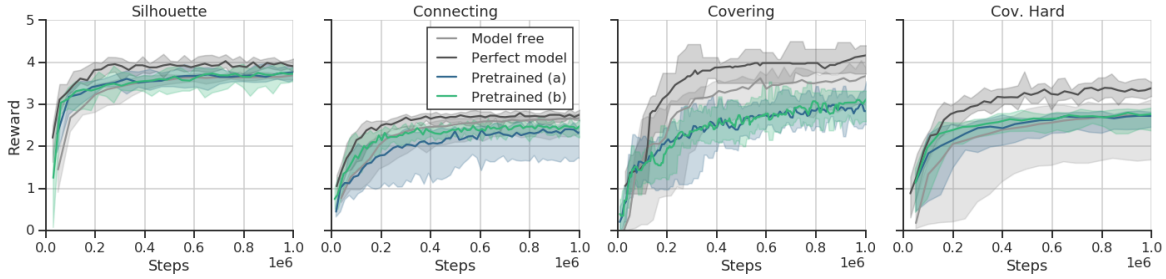

Figure E.2. Performance of agents with pre-trained models at small number of steps. *Pretrained (a)* uses a model pre-trained on a model free agent (the continuation of the light grey curve), while *Pretrained (b)* uses a model pre-trained on a model based agent (the continuation of the dark grey curve). Observe how the pre-trained curves match the perfect model curves at short times. All these curve use a same learning rate of $2\ 10^{-4}$ for fair comparison on short timescales. In all plots, solid lines are median performance across seeds and shaded regions extend between min and max seeds.

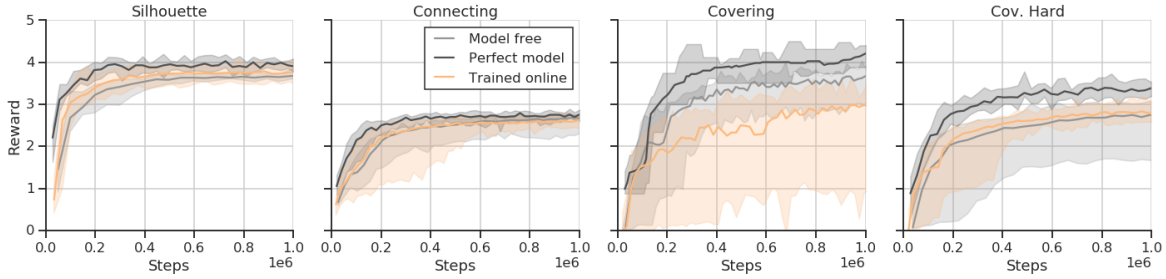

Figure E.3. Performance of agents learning a model of the environment at small number of steps. The model based agent use an MCTS budget of 5 that worked best in this setting. All these curve use a same learning rate of $2\ 10^{-4}$ for fair comparison on short timescales. In all plots, solid lines are median performance across seeds and shaded regions extend between min and max seeds.

F. Model-free continuous agent

F.1. RS0

We use a Retraced Stochastic Value Gradients (RS0, Heess et al. (2015); Munos et al. (2016); Riedmiller et al. (2018)) off-policy agent, with a shared observation pre-processor and independent actor and critic core models (MLP or GN). The critic is conditioned on the actions by concatenating them with the input to the model core (either MLP input features of graph globals). The actor learns a Gaussian distribution over each of the actions by outputting parameters using a linear policy head, conditioned on the last layer of the MLP or output globals of the GN. We use a value of 0.98 for the discount and calculated the retrace loss on sequences of length 5.

F.2. Exploration

While the Gaussian policy noise is often sufficient as a source of exploration, due to the highly multi-modal nature of placing objects, we injected additional fixed ϵ exploration by sampling a continuous action uniformly over the action range with probability ϵ, and sampling from the Gaussian otherwise. We set $\epsilon = .08$ for tasks with shorter episodes (*Silhouette* and *Covering Hard*) and $\epsilon = .03$ otherwise.

F.3. Dynamic curriculum

Due to the slower training of RS0 compared to DQN, and the large variance in learning time across the different configurations, we use a dynamic curriculum, only allowing agents to progress through the curriculum once they had achieved a certain performance in the current level.

The criteria for progressing through the curriculum is to obtain at least 50% of the maximum reward in at least 50% of the episodes (*Silhouette*, *Covering Hard*) or at least 25% of the maximum reward in at least 25% of the episodes (*Connecting*,

Covering). The threshold values were selected to ensure that the great majority of seeds would reach the maximum level of the curriculum during the allocated experiment time.

To avoid agents from progressing in the curriculum by just solving a particular type of scene, this criteria is applied independently over groups of episodes partitioned based on unique combinations of: number of targets, maximum target height, number of obstacles and maximum obstacle height, and using statistics from the last 200 episodes in each group.

F.4. Distributed setup

We run every experiment with 10 independent seeds, each of them with 8 actors, 1 learner and 1 FIFO replay (capacity=10^5 sequences). Additionally, an evaluation actor with the exploration disabled (and that did not feed data into the replay) is used to generate data to evaluate the dynamic curriculum criteria, and to monitor the overall performance in the task at maximum difficulty.

Due to the off-policy character of the algorithm, we did not set any synchronization between the actors generating the data and the learner obtaining batched of data from the replay. As a consequence, the relative number of actor steps per second and learner steps per second can vary drastically across the different architectures, depending of the relative speed differences between the forward pass (actors), and the backward pass (learner) of the models. Instead, we decided to use a wall-time criteria for terminating our experiments, stopping all experiments after one week of training, or after performance started decreasing.

G. Videos

Table G.1. Links to videos demonstrating constructing behavior for the best agents (as determined by evaluating on 10K episodes) on 10 random episodes. These episodes were taken from the hardest levels of the curriculum for each of the tasks, including Generalization (Gen.) settings where available. For each task, we used the same set of 10 random episodes across all agents to enable easy comparison. All videos are also provided here: https://tinyurl.com/y7wtfen9 .

Task	Best absolute agent	Best non-GN relative agent	Best relative agent	Best model-based relative agent
Silhouette	GN-RS0	RNN-RS0	GN-DQN	GN-DQN with MCTS at test time
Gen. 16 Blocks	GN-RS0	CNN-RS0	GN-DQN	GN-DQN with MCTS at test time
Connecting	RNN-RS0	RNN-RS0	GN-DQN	GN-DQN with MCTS at test time
Gen. Diff. Locs.	RNN-RS0	RNN-RS0	GN-DQN	GN-DQN with MCTS at test time
Gen. 4 Layers	RNN-RS0	RNN-RS0	GN-DQN	GN-DQN with MCTS at test time
Covering	GN-RS0	CNN-RS0	GN-DQN	GN-DQN with MCTS at test time
Covering Hard	GN-RS0	RNN-RS0	GN-DQN	GN-DQN with MCTS at train and test time

H. Further examples of scenes and constructions

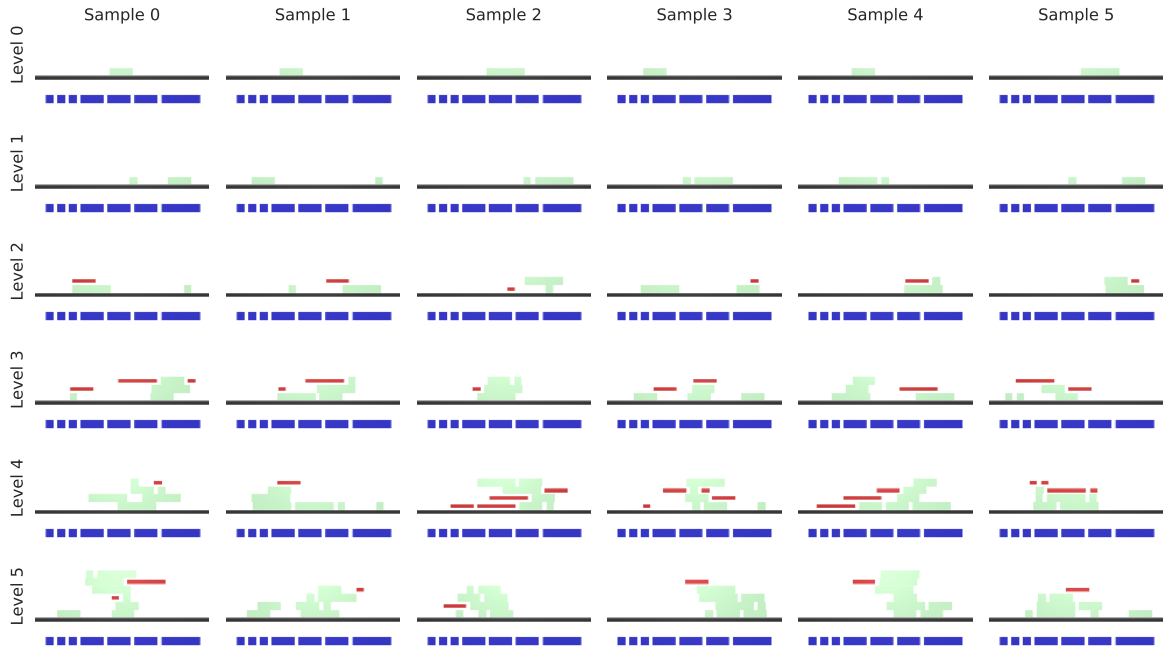

Figure H.1. Samples of the hardest scenes for the *Silhouette* task at each level of the curriculum. The n-th level of the curriculum consists of scenes uniformly sampled from the rows up to the n-th row. Hardest scenes for this task correspond to the n-th row.

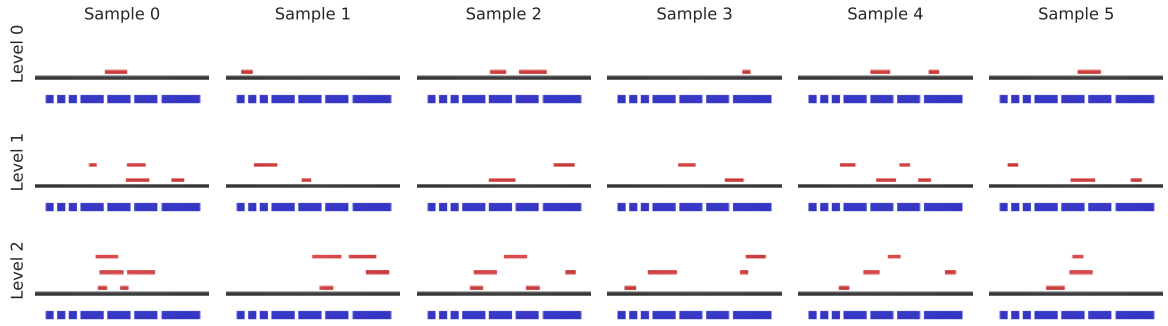

Figure H.2. Samples of the hardest scenes for the *Covering* task at each level of the curriculum. The n-th level of the curriculum consists of scenes uniformly sampled from the rows up to the n-th row. Hardest scenes for this task correspond to the n-th row.

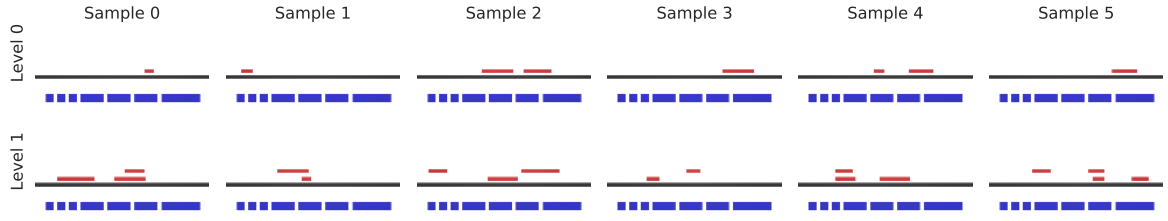

Figure H.3. Samples of the hardest scenes for the *Covering Hard* task at each level of the curriculum. The n-th level of the curriculum consists of scenes sampled from the rows up to the n-th row. Hardest scenes for this task correspond to the n-th row.

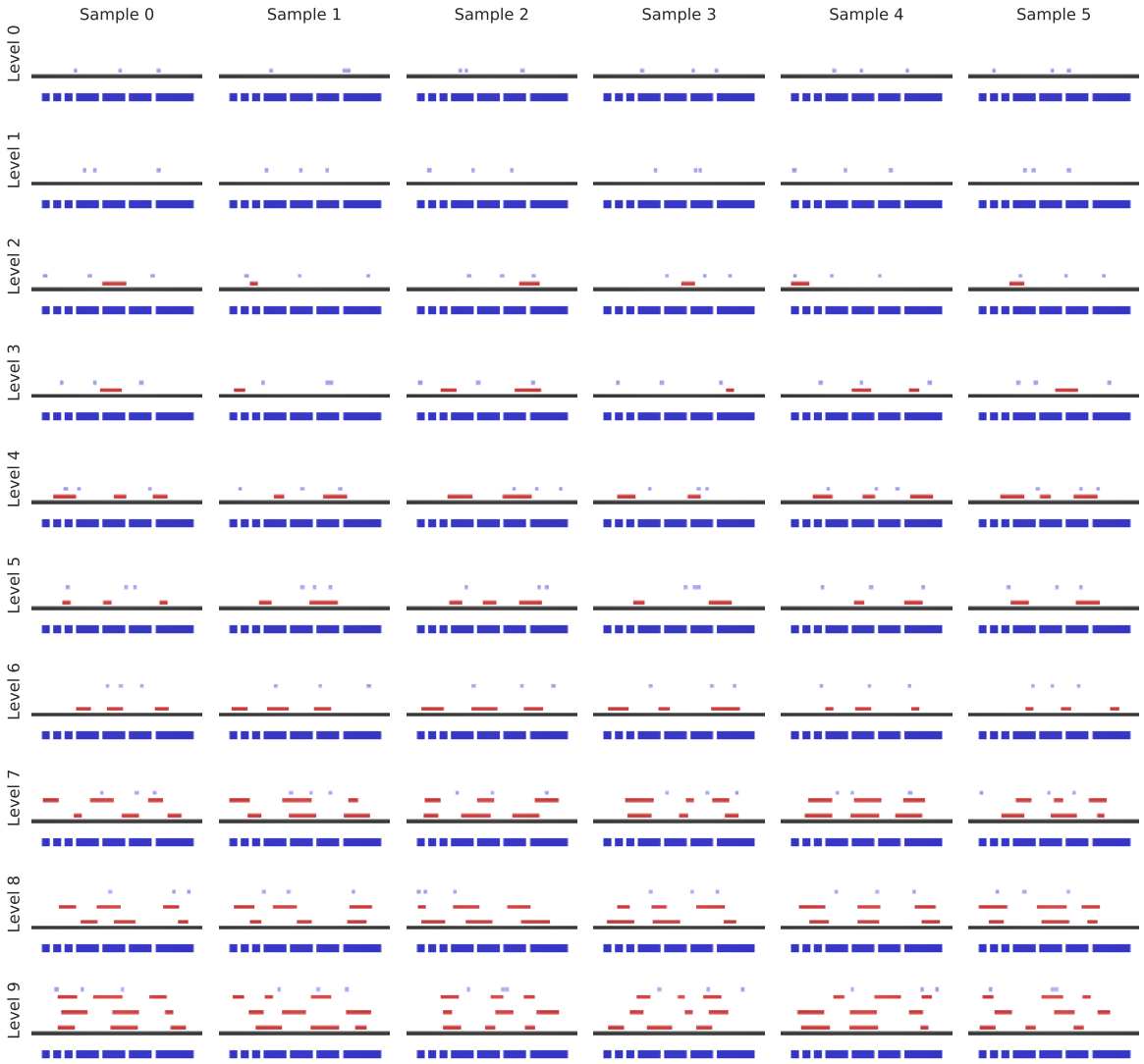

Figure H.4. Samples of the hardest scenes for the *Connecting* task at each level of the curriculum. The n-th level of the curriculum consists of scenes uniformly sampled from the rows up to the n-th row. Hardest scenes for this task correspond to the n-th row.

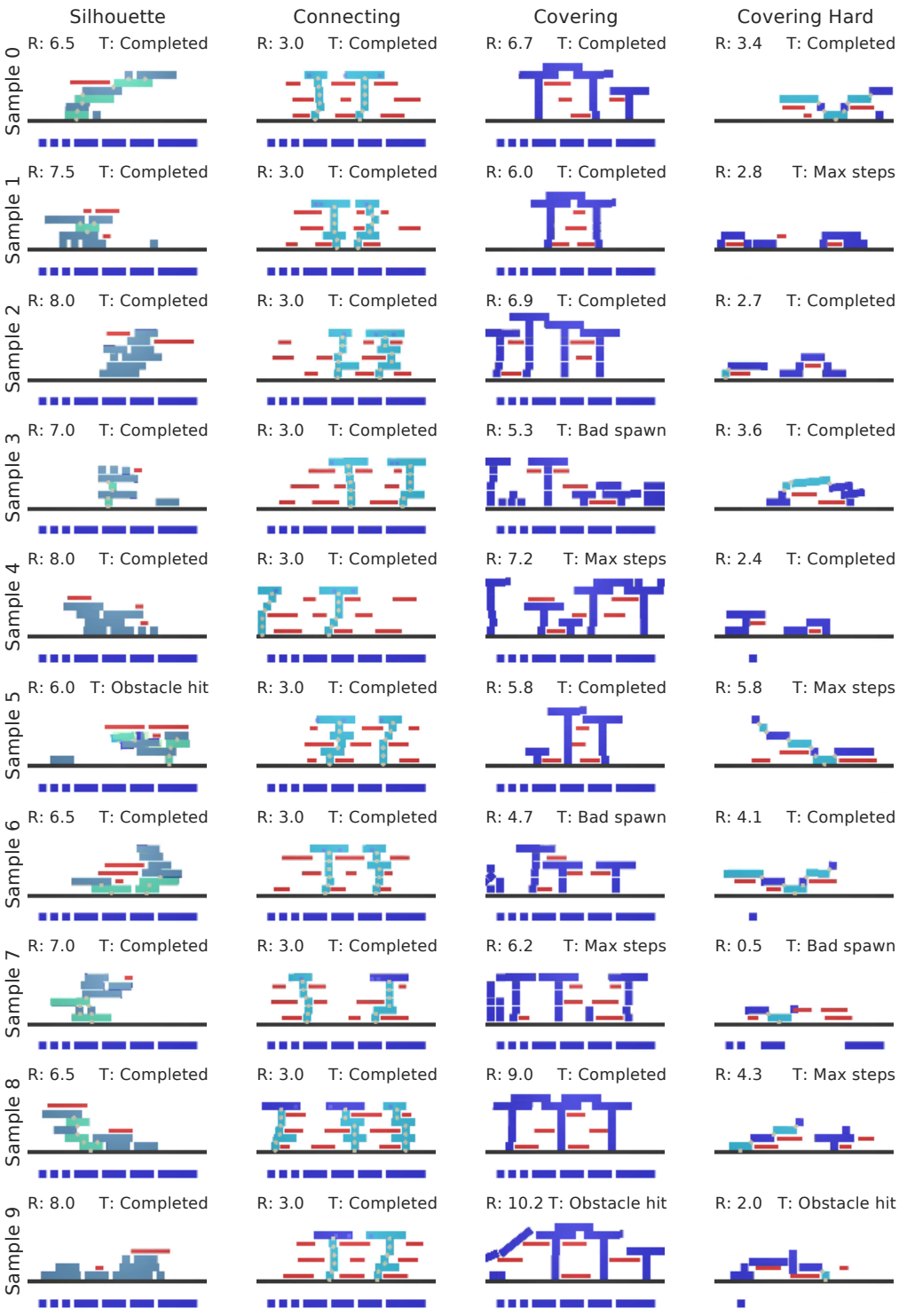

Figure H.5. Structures built by the Model-Based agent chosen randomly from the maximum difficulty levels of each task. The total episode reward (R), and termination reason (T) are displayed on top of each scene. Termination reasons are: *Completed* when the agent collects all of the reward in the task, *Max steps* when the agent hits reaches the maximum number of steps or runs out of blocks, *Obstacle hit* when a block hits an obstacle, *Bad spawn* when the agent places a block at a location that makes the new block overlap with an existing block and *Wrong edge* when the DQN agent chooses an edges that is not connecting an available block with a block in the scene.

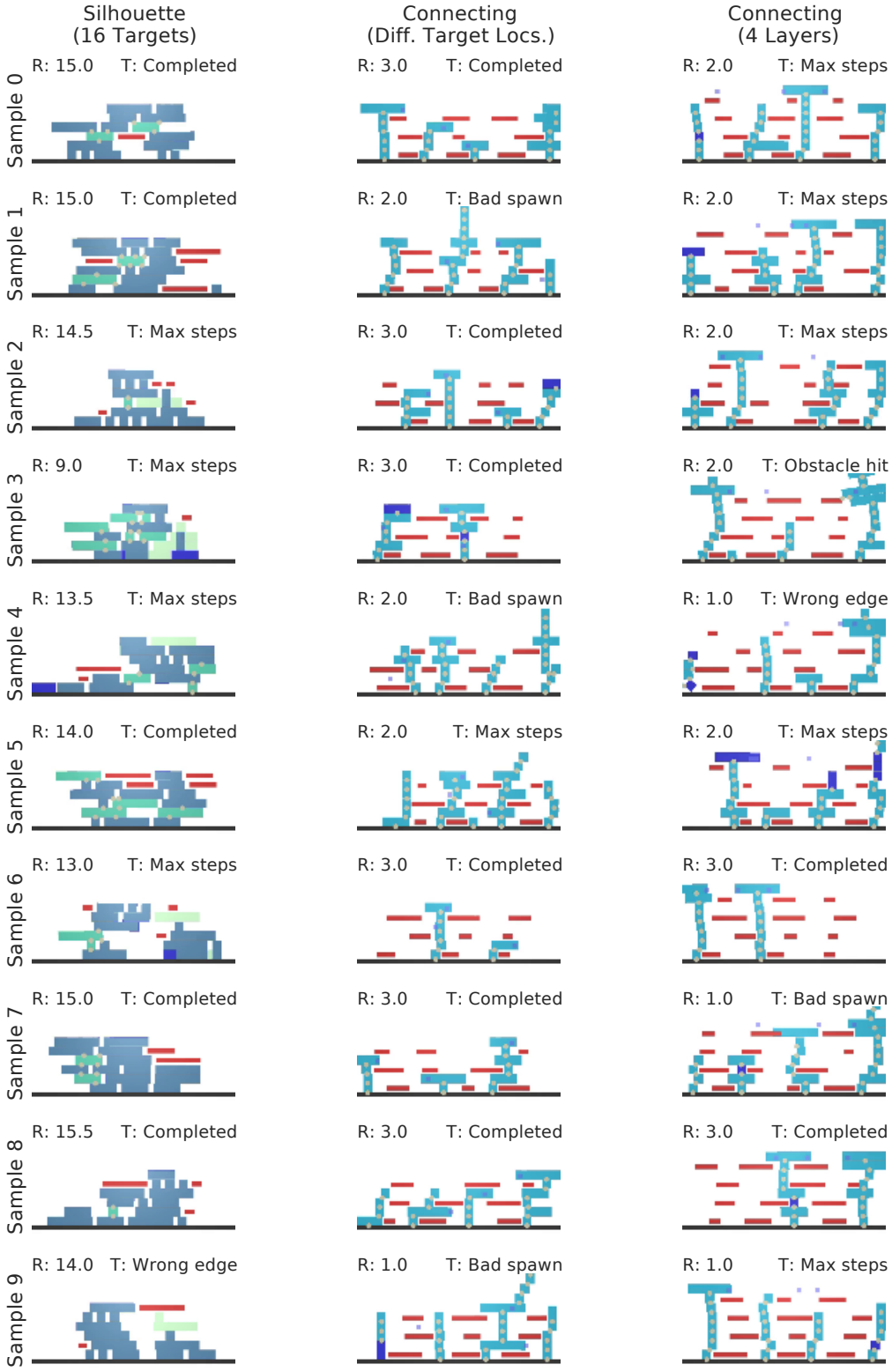

Figure H.6. Structures built by the Model-Based agent in generalization settings of the *Silhouette* and *Connecting* tasks. The total episode reward (R), and termination reason (T) are displayed on top of each scene as in Fig. H.5.