A L-AMIGo algorithm

Algorithm S1 Asynchronous learning step for L-AMIGo

Input: student batch size B_S, teacher policy batch size B_{policy}, grounding network batch size B_{ground}
$\mathcal{B}_{\text{policy}} \leftarrow \varnothing, \mathcal{B}_{\text{ground}} \leftarrow \varnothing$ ▷ Init teacher batches
$\mathcal{G} \leftarrow \varnothing$ ▷ Track descriptions seen thus far
while not converged **do**
    Sample batch $\mathcal{B}$ of size B_S from actors
    Train student on $\mathcal{B}$
    Add new descriptions ℓ_t in the batch to $\mathcal{G}$
    Update $\mathcal{B}_{\text{policy}}$ with (s_0, ℓ_t, r_t^T) tuples where goal ℓ_t completed ($r_t^T = +\alpha$) or episode ended ($r_t^T = -\beta$)
    if $|\mathcal{B}_{\text{policy}}| > B_{\text{policy}}$ **then**
        Train teacher policy on $\mathcal{B}_{\text{policy}}$; $\mathcal{B}_{\text{policy}} \leftarrow \varnothing$
    end if
    Update $\mathcal{B}_{\text{ground}}$ with (s_0, ℓ_{1st}) tuples
    if $|\mathcal{B}_{\text{ground}}| > B_{\text{ground}}$ **then**
        Train grounding net on $\mathcal{B}_{\text{ground}}$; $\mathcal{B}_{\text{ground}} \leftarrow \varnothing$
    end if
end while

Algorithm S1 describes the joint student/teacher learning step of L-AMIGo. Batches of experience $\mathcal{B}$ are generated from actors, which are used to update the student policy, and the teacher policy at different intervals defined by batch sizes B_{policy} and B_{ground}. The set of goals $\mathcal{G}$ known to the teacher is also progressively updated.

To reiterate the teacher training process: the teacher policy network is trained on tuples of student initial state s_0, proposed goal ℓ_t, and teacher rewards r_t^T (which is $+\alpha$ if the goal was completed by the student in $\geq t^*$ steps, or $-\beta$ if the goal was completed in $< t^*$ steps or never completed before episode termination). The teacher grounding network is trained on tuples of initial state s_0 and the first language description encountered along that trajectory ℓ_{1st}. Given s_0, the grounding network is asked to predict 1 for ℓ_{1st} and 0 for all other goals known to the teacher at the time.

B Details on naive message reward and equivalencies to prior work

As discussed in Section 2, prior approaches to language-guided reward shaping assume either a single extrinsic goal, or that language annotations are always helpful for progress in the environment. Here we show how these methods can all be approximately captured by a naive message reward, which our experiments show is insufficient for learning in sufficiently large linguistic spaces (Figure 3).

- Harrison et al. [24] (also [51]) propose a policy shaping method that generates language annotations for (action, state) pairs from simulated oracles, then for each language annotation learns a "critique policy" that mimics the human action distributions for the current state and language annotation. This is used to update the prior action distribution of the learned policy during training: given a state, the most likely language annotation over all possible actions is inferred (via Bayes' rule); then the critique policy for this language annotation is mixed in with the current policy's action distribution via a tunable hyperparameter. While the focus is on policy shaping rather than reward shaping, agents are simply pushed in the direction of the most salient language annotation, and there are no preferences for or against certain language. The overall effect, then, is that policies are pushed indiscriminately towards encountering *any* message in the environment, which is similar to a fixed message reward.

  A further limitation of Harrison et al. [24] that prevents straightforward application here is that it requires sufficient training data for training a critique policy offline *for every possible message encountered in the environment*. In contrast, we assume no such pretraining data; in fact, we have no knowledge about what messages we might encounter at all.

- ELLA [31] is closer to our setting, since the authors acknolwedge that not all messages or subgoals in a trajectory may be relevant for an overall goal, especially in the multi-task settings they explore where the extrinsic goal is subject to change. Accordingly, ELLA proposes to learn a "Relevance Classifier" that predicts whether or not a message is useful for the extrinsic goal by training on the messages encountered along trajectories that resulted in positive reward. However, in our setting, we have no extrinsic goals, and in most of our tasks, random exploration fails to attain *any* positive trajectories to provide such training data for a classifier (Figure 3, IMPALA curves). We could implement ELLA with an uninformative reference classifier, i.e. one that assumes all messages are relevant for the extrinsic goal. Then ELLA reduces to simply giving a fixed reward for any message encountered.
- Finally, a large class of reward shaping and inverse RL methods operate primarily by giving rewards associated with a (linguistic) extrinsic goal [3, 5, 21–24, 31, 44, 51, 53]. As one representative example, LEARN [22] proposes to train a model to associate an action at a given timestep with the probability it is *related* to an extrinsic language command: $p_R(a_t, \ell)$, as well as the complementary probability that an action is *unrelated*: $p_U(a_t, \ell) = 1 - p_R(a_t, \ell)$. The difference in these probabilities can thus be used as a reward signal: $r_t^i = p_R(a_t, \ell) - p_U(a_t, \ell)$.

  However, in our case we have no extrinsic instruction; rather, we have many intermediate low-level messages of unknown relevance to the extrinsic goal. A naive solution for LEARN-style approaches is thus to do reward shaping for every intermediate message, i.e. assign rewards whenever any annotation ℓ is deemed relevant. Moreover, we do not need to predict when an action is relevant to ℓ; the frequent annotations we receive already confirm that an action is linguistically relevant. Thus, in our setting, we can set $p_R(a, \ell) = 1$ if ℓ is observed in the given state, and $p_R(a, \ell) = 0$ otherwise.[9] Using this as the reward is equivalent to the baseline employed here. Technically, LEARN proposes a slightly modified reward $r_t'^i = \gamma r_t^i - r_{t-1}^i$, i.e. the difference in reward between timesteps where γ is the MDP discount factor, but this made no difference in our experiments; thus, for brevity we report the single fixed reward.

For the naive reward baseline reported in Figure 3, we performed a grid search on the intrinsic reward coefficient $\lambda \in \{0.1, 0.5, 1.0\}$, though modifying λ made no difference and results are reported with $\lambda = 0.1$. In all cases, agents trained with such rewards fail because without some notion of message novelty or difficulty, the agents are stuck exploiting easy-to-achieve, locally-optimal messages (e.g. running into the nearest wall in MiniHack, or the nearest door in MiniGrid); in fact, rewards for easy messages *discourage* exploration and lead to worse reward than even the vanilla IMPALA baselines! Thus, some way of measuring novelty/progress in the space of language annotations must be used, which is operationalized in L-AMIGo (via the continually growing difficulty threshold given to the teacher) and L-NovelD (via the decaying reward given for seeing the same message over and over again), though note that for L-NovelD, using a message reward with novelty-based decay is insufficient (Appendix F.2).

C Architecture and training details

Here, we describe the architecture, training details, and hyperparameters for the MiniGrid and MiniHack tasks. Our code is available in the supplementary material and also at `https://anonymized`.

C.1 MiniGrid

All models are adapted from Campero et al. [7]. Figure S1 from Campero et al. [7] details the architecture of the standard AMIGo student and teacher.

AMIGo student. The student gets an $7 \times 7 \times 3$ partial and egocentric grid representation, where each cell in the 7×7 grid is represented by 3 features indicating the type of the object, color of

[9] An even stricter interpretation would be to give partial rewards even for states with the null message $\varnothing$ by training a classifier to predict relevance for any non-null linguistic state. However, intermediate message rewards are already fairly common in our setting (it is easy to walk to the nearest wall in MiniHack, for example), and this would not solve the fundamental problem that without learning to disprefer certain messages, an agent will be stuck pursuing locally optimal messages.

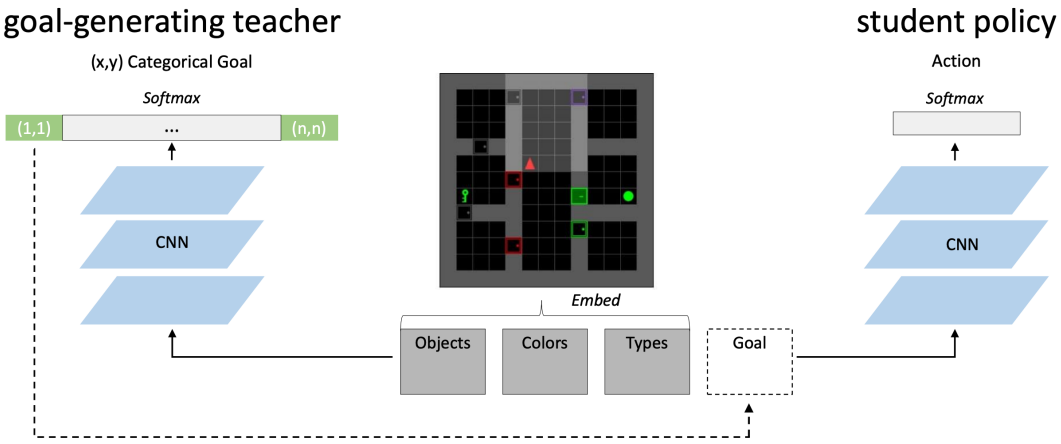

Figure S1: **Original AMIGo model overview.** Original figure from Campero et al. [7], reproduced with permission.

object, and object state (e.g. for doors, open/closed). The student first embeds the features into type/color/state embeddings of size 5, 3, and 2 respectively, as well as the (x, y) goal as a singleton feature, to get a $7 \times 7 \times 11$ grid which is then fed through a 5-layer convolutional neural network interleaved with Exponential Linear Units (ELUs; 11). Each layer has 32 output channels, filter size of $(3, 3)$, 2 stride, and 1 padding. The output of the ConvNet is then flattened and fed through a fully-connected layer to produce a 256-d state embedding. The policy and value functions are linear layers on top of this state embedding. The policy in particular produces a distribution over 7 possible actions (forward, pick up, put down, open, left, right, and a "done" action).

AMIGo teacher. The teacher gets an $N \times N \times 3$ fully observed grid encoding of the environment, where N varies according to the environment size. Like the student, the teacher embeds the grid representation into embeddings which are then fed through a 4-layer dimensionality-preserving neural network again interwoven with ELUs. Each convolutional layer has 16 output channels, filter size of $(3, 3)$, 1 stride, and 1 padding, except the last layer which has only 1 output channel. The ConvNet processes the input embeddings into a spatial action distribution over grid locations, from which an goal is sampled. The value head takes as input the penultimate layer from the ConvNet (i.e. a grid of $N \times N \times 16$) to produce the value estimate.

L-AMIGo Student. The L-AMIGo student has the same ConvNet base as the standard AMIGo student, but without an (x, y) goal channel feature (so the input to the ConvNet is $7 \times 7 \times 10$ instead of 11). To encode the goal, the student uses a one-layer Gated Recurrent Unit (GRU; Cho et al. [10]) recurrent neural network (RNN) with embedding size 64 and hidden size 256. The last hidden state of the GRU is taken as the goal representation, which is then concatenated with the state embedding to be fed into the policy and value functions, respectively.

L-AMIGo Teacher. The L-AMIGo *policy network* has the same ConvNet as the standard AMIGo teacher, but without the last layer (so the output is a $N \times N \times 16$ grid). The last output of the ConvNet is averaged across the spatial map to form a final 16-dimensional state embedding. The L-AMIGo teacher also has a GRU of same dimensionality as the L-AMIGo student. To propose a goal, the teacher embeds each known goal in the vocabulary with the GRU to produce 256-dimensional goal embeddings which are then projected via a linear layer into the 16-dimensional goal embedding space. The dot product of the goal embeddings and the state form the logits of the goal distribution. The value head takes as input the $N \times N \times 16$ unaveraged state representation concatenated with the logits of the distribution of language goals.

The *grounding network* also uses the 16-dimensional state embedding and the same GRU as the L-AMIGo policy network, but the 256-d goal embeddings are projected via a separate linear layer to a separate set of 16-dimensional embeddings. The dot product between these grounder-specific goal embeddings and the state embedding represent the log probabilities of the goal being achievable in an environment.

NovelD and L-NovelD. For NovelD experiments we use the student policy of L-AMIGo but with the goal embedding always set to the 0 vector.

The NovelD RND network uses the same convolutional network as the AMIGo/L-AMIGo students, taking in an egocentric agent view. The L-NovelD message embedding network uses a GRU parameterized identically to those of the L-AMIGo student and teacher. For NovelD experiments we set $\alpha = 1$, scale the RND loss by 0.1, and use the same learning rate as the main experiments. Using a grid search, we found optimal scaling factors of the standard NovelD reward to be 0.5 and the L-NovelD reward (i.e. λ_ℓ) also to be 0.5.

Hyperparameters. We use the same hyperparameters as in the original AMIGo paper: a starting difficulty threshold t^* of 7, a maximum difficulty t^* of 100, positive reward for the teacher $+\alpha = 0.7$, negative reward $-\beta = -0.3$, learning rate 10^{-4} which is linearly annealed to 0 throughout training, batch size 32, teacher policy batch size 32, teacher grounder batch size 100, unroll length 100, RMSprop optimizer with $\varepsilon = 0.01$ and momentum 0, entropy cost 0.0005, generator entropy cost 0.05, value loss cost 0.5, intrinsic reward coefficient $\lambda = 1$.

C.2 MiniHack

Models are adapted from baselines established for the NetHack [28] and MiniHack [41] Learning Environments.

AMIGo student. The student is a recurrent LSTM-based [25] policy. The NetHack observation contains a 21×79 matrix of glyph identifiers, a 21-dimensional feature vector of agent stats (e.g. position, health, etc), and a 256-character (optional) message. The student produces 4 dense representations from this observation which are then concatenated to form the state representation.

First, an embedding of the entire game area is created. Each glyph is converted into a 64-dimensional vector embedding, i.e. the input is now a $21 \times 79 \times 64$ grid. This entire grid is fed through a 5-layer ConvNet interleaved with ELUs. Each conv layer has a filter size of $(3, 3)$, stride 1, padding 1, and 16 output channels, except for the last which has 8 output channels. **Second**, an embedding of a 9×9 egocentric crop of the grid around the agent is created. This is created by feeding the crop through another separately-parameterized ConvNet of the same architecture. **Third**, the 21-d feature vector of agent stats is fed into a 2-layer MLP (2 layers with 64-d outputs with ReLUs in between) to produce a 64-d representation of the agent stats. **Fourth**, the message is parsed with the BERT [16] WordPiece tokenizer and fed into a GRU with embedding size 64 and hidden size 256. These four embeddings are then concatenated and form the state representation, which is then fed into the LSTM which contextualizes the current representation. This final representation is fed into linear policy and value heads to produce action distribution and value estimates.

AMIGo teacher. The teacher first embeds the agent stats using the same MLP that the student uses. It has the same 5-layer ConvNet used by the student to embed the full game area, except the input channels are modified to accept the 64-d feature vector concatenated to every cell in the 21×79 grid, so the input to the ConvNet is $21 \times 79 \times 128$. Additionally, the output layer has only 1 output channel. This produces a spatial action distribution over the 21×79 grid from which a "goal" as (x, y) coordinate is sampled.

L-AMIGo student. The student works identically to the standard AMIGo student, except without the teacher goal channel concatenated into the grid input. Also, in addition to the 4 representation constructed from the observation, the student gets 2 more representations constructed from the teacher's language goal. **First**, the student uses the same GRU used to process the game message to encode the teacher's language goal to create a 256-d representation. **Second**, the difference between the observed language embedding and the teacher language embedding is used as an additional feature (when this is the 0 vector then the student has reached the goal). All together, this forms 6 representations that together constitute the state representation.

L-AMIGo teacher. The teacher constructs the same state representation as the standard AMIGo student (*not* the AMIGo teacher). The teacher then embeds all known goals with a word-level GRU with the same architecture as the message network used by the student. These 256-d message embeddings are then projected to the state representation hidden size, and the dot product between

message embeddings and state representations forms the distribution over goals. Similarly, for the grounding network, the 256-d message embeddings are projected via a separate linear layer to produce probabilities of achievability. Like in MiniGrid, the state representation and the goal logits are used as input for the value head.

NovelD and L-NovelD. As in MiniGrid experiments, for NovelD experiments we use the L-AMIGo student policy where the goal embedding is always 0.

The NovelD RND network uses the cropped ConvNet representation of the student, fed through a linear layer to produce a 256-d state representation. Egocentric crops change more over time and are thus a more reliable signal of "novelty" than the full grid representation (verified with experiments). Additionally, as done in Burda et al. [6], two more additional layers are added to the final MLP of the predictor network only (not the random target network). The L-NovelD message RND network uses the same word-level GRU architecture of the student. We set $\alpha = 0.5$, scale the RND loss by 0.1, and use the same learning rate as the main student policy. The standard NovelD reward is left alone, and the L-NovelD reward hyperparameter (λ_ℓ) is grid-searched and set to 50 for all MiniHack experiments except Quest-{Easy,Medium}, where it is 30.

Hyperparameters. We generally stick to the same hyperparameters of Samvelyan et al. [41]. We use a starting difficulty threshold t^* of 1, a maximum difficulty t^* of 2 (a high goal difficulty is not as important for MiniHack), positive reward for the teacher $+\alpha = 0.7$, negative reward $-\beta = -0.3$, linearly-annealed learning rate 10^{-4}, batch size 32, teacher policy batch size 32, teacher grounder batch size 500, unroll length 100, RMSprop optimizer with $\varepsilon = 0.01$ and momentum 0, entropy cost 0.0005, generator entropy cost 0.05, value loss cost 0.5, intrinsic reward coefficient $\lambda = 0.4$. ε-greedy exploration was used for the teacher policy with $\varepsilon = 0.05$, which we found helped learning.

C.3 Compute Details

Each model was run for 5 independent seeds on a machine in an independent cluster with 40 CPUs, 1 Tesla V100 GPU, and 64GB RAM. Runs take between 4 hours (for ObstructedMaze_1Dl) to 20 hours (for the longest MultiRoom-N4-Extreme and KeyCorridorS5R3 tasks).

D Additional tables visualizations of main results

D.1 Full numeric tables

Table S1 contains full numbers for IQM performance. This is the same data as Figure 4, just summarized in numeric form.

Table S1: **Full IQM numbers.** IQM performance ($\pm$ 95% bootstrapped CIs) for models across tasks.

Environment	AMIGo	L-AMIGo	NovelD	L-NovelD
KeyCorridorS3R3	0.86 (0.77, 0.89)	0.89 (0.85, 0.90)	0.88 (0.87, 0.88)	**0.90** (0.83, 0.90)
KeyCorridorS4R3	0.82 (0.11, 0.90)	**0.89** (0.80, 0.91)	0.07 (0.02, 0.45)	**0.89** (0.87, 0.91)
KeyCorridorS5R3	0.92 (0.54, 0.93)	**0.93** (0.92, 0.93)	0.00 (0.00, 0.03)	0.88 (0.70, 0.93)
ObstructedMaze_1Dl	0.18 (0.13, 0.91)	**0.91** (0.89, 0.93)	0.23 (0.15, 0.64)	0.87 (0.39, 0.93)
ObstructedMaze_2Dlhb	0.61 (0.14, 0.86)	0.80 (0.18, 0.83)	0.86 (0.82, 0.88)	**0.89** (0.86, 0.90)
ObstructedMaze_1Q	0.17 (0.06, 0.86)	0.88 (0.77, 0.92)	**0.91** (0.83, 0.93)	**0.91** (0.91, 0.93)
River	0.47 (0.42, 0.49)	**1.00** (0.47, 1.00)	0.54 (0.52, 0.73)	**1.00** (0.58, 1.00)
WoD-Medium	**1.00** (1.00, 1.00)	**1.00** (0.82, 1.00)	0.00 (0.00, 0.40)	0.50 (0.00, 1.00)
WoD-Hard	0.00 (0.00, 0.00)	**0.85** (0.00, 0.90)	0.00 (0.00, 0.00)	0.75 (0.29, 0.92)
Quest-Easy	**1.00** (0.99, 1.00)	**1.00** (0.00, 1.00)	**1.00** (0.25, 1.00)	**1.00** (1.00, 1.00)
Quest-Medium	**1.00** (0.99, 1.00)	**1.00** (1.00, 1.00)	0.00 (0.00, 0.67)	0.97 (0.00, 1.00)
MultiRoom-N2-Extreme	0.25 (0.08, 0.69)	**0.82** (0.09, 0.95)	0.64 (0.46, 0.84)	0.81 (0.41, 0.96)
MultiRoom-N4-Extreme	0.00 (0.00, 0.00)	**0.01** (0.01, 0.58)	0.00 (0.00, 0.01)	0.00 (0.00, 0.24)
MiniGrid	0.62 (0.47, 0.81)	0.88 (0.78, 0.89)	0.51 (0.47, 0.59)	**0.89** (0.81, 0.90)
MiniHack	0.53 (0.51, 0.59)	**0.81** (0.65, 0.91)	0.31 (0.23, 0.41)	0.65 (0.46, 0.79)
Overall	0.57 (0.50, 0.66)	**0.84** (0.74, 0.89)	0.41 (0.35, 0.47)	0.76 (0.66, 0.83)

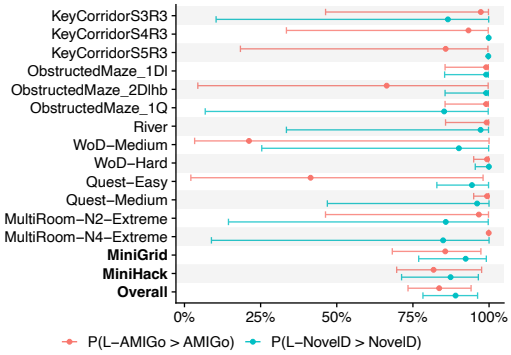

Figure S2: **Probability of improvement.** Probability of improvement of L-AMIGo over AMIGo, and L-NovelD over NovelD, as measured by Mann-Whitney U tests between their final performances. Plot elements same as Figure 4.

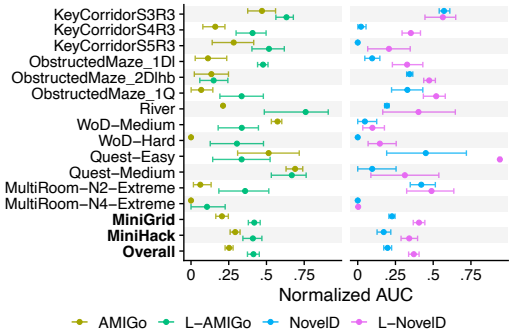

Figure S3: **Normalized AUC.** Normalized area under the curve (AUC) for AMIGo, L-AMIGo, NovelD, and L-NovelD. Plot elements same as Figure 4.

D.2 Probability of improvement

In addition to the IQM, another alternative interpretation of results as advocated in Agarwal et al. [2] is to use the *probability of improvement* of algorithm A over algorithm B, as measured by the nonparametric Mann-Whitney U test between independent runs from both algorithms. Figure S2 shows results from such an evaluation, again evaluated over bootstrapped confidence intervals constructed from 5000 samples per model/env combination. Qualitative results are the same as in Figure 4: overall, across environments, L-AMIGo and L-NovelD are both highly likely to outperform AMIGo and NovelD.

D.3 Area Under the Curve (AUC).

The IQM (Figure 4) and probability plots (Figure S2) use point estimates of ultimate performance attained after a fixed compute budget, which does not measure differences in sample efficiency between runs. As a measure which also elucidates differences in sample efficiency, we to use the normalized Area Under the Curve (AUC) to compare runs, as used in prior work [22]. Results are in Figure S3, and are qualitatively similar to the IQM and probability plots, but here, differences in sample efficiency can be seen in some environments, e.g. in KeyCorridorS3R3 for L-AMIGo, which are absent from the IQM plot.

E MiniGrid experiments with language encoded into the state representation

Here we run MiniGrid models with language encoded into the state representation. Full training curves for MiniGrid tasks only are located in Figure S4, IQM summary statistics for all environments (with only MiniGrid environments changed) are located in Figure S5, and Table S2 contains updated raw performance numbers. As discussed in the main text, recall that MiniHack models all already encode language into the state representation. We make the following observations about how these experiments differ from those in the main text:

Incorporating language into the feature space improves performance across all models. However, just incorporating language in the feature space is insufficient for competitive performance alone: while the IMPALA baseline is able to make more progress in simpler environments (e.g. now nearly solving KeyCorridorS3R3), it still lags behind and is unable to solve the harder environments. This clearly illustrates that in order to reap the benefits of language, it is important to use it in conjunction with exploration, not just as a feature.

L-AMIGo continues to outperform AMIGo, though the differences are smaller. While the tables show that both L-AMIGo and AMIGo can solve each task, reaching roughly similar final performance, they differ in sample efficiency as well as training stability (Figure S4, as well as the smaller error bars of L-AMIGo in Figure S5). Similar to the results in the main text, the full training

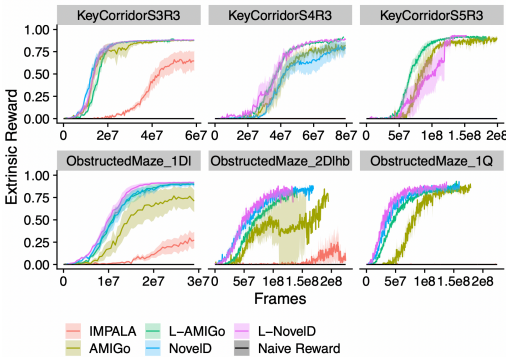

Figure S4: **Training curves for MiniGrid with language states**. Plot elements same as Figure 3.

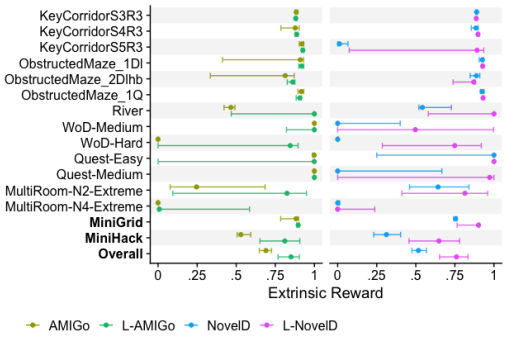

Figure S5: **IQMs with language states**. Plot elements same as Figure 4.

curves in Figure S4 show that while L-AMIGo and AMIGo reach similar asymptotic performance, L-AMIGo learns quicker or more stably. This happens quite clearly in KeyCorridorS5R3 and the Maze environments.

L-NovelD and NovelD performance on MiniGrid are similar. This is the case except on a few tasks: KeyCorridorS4R3, and KeyCorridorS5R3 where NovelD is still unable to learn. Note that these NovelD results are somewhat unsurprising given the L-NovelD ablations and discussion in Appendix F.2; as we discuss there, L-NovelD is simply an adaptation of NovelD that enables a more precise tradeoff between language and state-based novelty. Similarly to how L-NovelD did not significantly outperform NovelD with language in the RND representation in Appendix F.2, L-NovelD does not outperform NovelD here, showing that for MiniGrid it is sufficient to naively combine the language and state representations. However, the MiniHack results in the main text indicate that there are settings where it is beneficial to have the separate L-NovelD term, and insufficient to solely encode language into the state representation.

To conclude, while adding language to the state representation results in smaller and more subtle performance differences in MiniGrid environments, the main conclusion (that language variants outperform their non-linguistic baselines) remains unchanged. As the last row of Table S2 shows, overall, L-AMIGo outperforms AMIGo by 23% (.16 absolute), and L-NovelD outperforms NovelD by 46% (.24 absolute) across environments; both differences remain statistically significant.

Table S2: **IQM numbers for MiniGrid with language states.** Table elements same as Table S1.

Environment	AMIGo	L-AMIGo	NovelD	L-NovelD
KeyCorridorS3R3	0.88 (0.88, 0.89)	0.88 (0.88, 0.89)	**0.89 (0.88, 0.89)**	**0.89 (0.88, 0.89)**
KeyCorridorS4R3	0.88 (0.79, 0.90)	0.89 (0.88, 0.89)	0.89 (0.85, 0.90)	**0.90 (0.89, 0.91)**
KeyCorridorS5R3	0.92 (0.90, 0.93)	**0.93 (0.92, 0.93)**	0.01 (0.00, 0.07)	0.89 (0.07, 0.93)
ObstructedMaze_1Dl	0.91 (0.41, 0.93)	0.92 (0.90, 0.93)	**0.93 (0.91, 0.93)**	**0.93 (0.92, 0.93)**
ObstructedMaze_2Dlhb	0.81 (0.34, 0.87)	0.86 (0.83, 0.87)	**0.89 (0.85, 0.91)**	0.87 (0.74, 0.88)
ObstructedMaze_1Q	0.92 (0.90, 0.93)	0.91 (0.88, 0.92)	0.92 (0.91, 0.94)	**0.93 (0.93, 0.94)**
MiniGrid	0.88 (0.79, 0.90)	**0.90 (0.89, 0.90)**	0.75 (0.74, 0.76)	**0.90 (0.76, 0.91)**
Overall (w/ MiniHack)	0.69 (0.65, 0.73)	**0.85 (0.77, 0.90)**	0.52 (0.47, 0.57)	**0.76 (0.65, 0.83)**

F Ablations

F.1 L-AMIGo grounding network

Figure S6 shows performance of L-AMIGo without the grounding network, where the policy network directly produces a distribution over goals without first predicting goal achievability, aggregated across domains. Overall, L-AMIGo without the grounding network performs reasonably well, matching full L-AMIGo performance on MiniHack. On MiniGrid, we see a modest difference in aggregate performance, though importantly, we also see greatly increased training stability with the grounding network

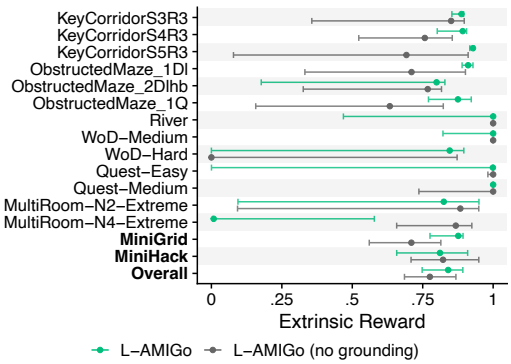

Figure S6: **L-AMIGo grounding network ablation**. IQM of L-AMIGo with and without grounding network across environments. Plot elements same as Figure 4.

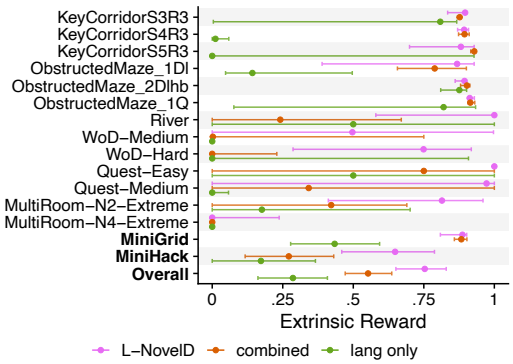

Figure S7: **L-NovelD ablations**. IQMs for Full L-NovelD, NovelD with combined state/language input, and language-only L-NovelD across environments. Plot elements same as Figure 4.

on individual environments. Specifically, the confidence intervals are much larger for L-AMIGo without the grounding network on **KeyCorridorS{3,4,5}R3** and **ObstructedMaze_{1Dl,1Q}**.

We hypothesize that the difference between MiniGrid and MiniHack tasks is because MiniGrid goals differ more between episodes: for example, since the colors of the doors are randomly shuffled, not all environments have red doors, so it is helpful to explicitly predict such features of the environment with a grounding network. In contrast, MiniHack envs have a more consistent set of goals (e.g. every WoD seed has a wand, a minotaur, etc), and so the grounding network is less necessary in this case.

F.2 L-NovelD components

To examine the relative performance of each component of L-NovelD, we run ablation experiments by (1) using the language-based reward *only*, or (2) combining the language and state embedding into a single representation. As discussed in Appendix B, using the language-based reward only is equivalent to the naive message baseline (and thus prior work like [22, 24, 31]) with an RND-determined novelty-based decay.

Results are in Figure S7. They show that using the language reward alone results in uniformly worse performance across environments, suggesting that a naive message-based reward, even with novelty decay, underperforms and it is important to provide a simpler navigation-based bonus to encourage exploration.[10] Additionally, while combining the state and language into a single embedding works well for MiniGrid, it does not work as well for MiniHack tasks, suggesting that the additional flexibility afforded by the separate L-NovelD term can be helpful in many settings. In principle, it should be possible to tune the state and language embedding sizes of combined NovelD to see comparable performance to L-NovelD, but the point of L-NovelD is to clarify the contributions made by both the language and state and make it easier to trade-off between the two.

G Full description of tasks and language

Here we describe each task in MiniGrid and MiniHack in detail, and enumerate the list of messages available in each task. Examples of all tasks explored in this work are located in Figure S9.

G.1 MiniGrid

Language. As described in Section 6.1, the set of possible descriptions in the BabyAI language 652 involves *goto, open, pickup*, and *putnext* commands which can be applied to boxes, doors, and balls, optionally qualified by color. These messages can be grouped into 66 message "templates" where specific colors are replaced with a placeholder <C>, as shown in Figure S8. Not all messages

[10] Note also that message-only NovelD underperforms L-AMIGo as well.

```
go to the <C> ball
go to the <C> box
go to the <C> door
go to the <C> key
go to the ball
go to the box
go to the door
go to the key
open the <C> box
open the <C> door
open the box
open the door
pick up the <C> ball
pick up the <C> box
pick up the <C> key
pick up the ball
pick up the box
pick up the key
put the <C> ball next to the <C> ball
put the <C> ball next to the <C> box
put the <C> ball next to the <C> door
put the <C> ball next to the <C> key
put the <C> ball next to the ball
put the <C> ball next to the box
put the <C> ball next to the door
put the <C> ball next to the key
put the <C> box next to the <C> ball
put the <C> box next to the <C> box
put the <C> box next to the <C> door
put the <C> box next to the <C> key
put the <C> box next to the ball
put the <C> box next to the box
put the <C> box next to the door
put the <C> box next to the key
put the <C> key next to the <C> ball
put the <C> key next to the <C> box
put the <C> key next to the <C> door
put the <C> key next to the <C> key
put the <C> key next to the ball
put the <C> key next to the box
put the <C> key next to the door
put the <C> key next to the key
put the ball next to the <C> ball
put the ball next to the <C> box
put the ball next to the <C> door
put the ball next to the <C> key
put the ball next to the ball
put the ball next to the box
put the ball next to the door
put the ball next to the key
put the box next to the <C> ball
put the box next to the <C> box
put the box next to the <C> door
put the box next to the <C> key
put the box next to the ball
put the box next to the box
put the box next to the door
put the box next to the key
put the key next to the <C> ball
put the key next to the <C> box
put the key next to the <C> door
put the key next to the <C> key
put the key next to the ball
put the key next to the box
put the key next to the door
put the key next to the key
```

Figure S8: **Full list of possible MiniGrid messages**. Messages are synthesized with the BabyAI [9] grammar, then divided into 66 templates. <C> is one of 6 possible colors: *grey*, *green*, *blue*, *red*, *purple*, *yellow*.

are needed for success on MiniGrid tasks, nor achieved by expert policies during training. We explain which messages are needed and/or encountered for each task below.

KeyCorridorS{3,4,5}R3. In these tasks (Figure S9a–c), the agent is tasked with picking up a ball behind a locked door. To do so, it must first find and pick up the key which is hidden in (possibly several) nested rooms, return to the locked door, unlock the door, place the key down, and pick up the ball. The agent start location, object colors, and room/door positions are randomized across seeds.

A typical policy trained on each task will encounter anywhere from 92 (KeyCorridorS3R3) to 141 messages (KeyCorridorS{4,5}R3) throughout training. Regardless of the environment size, a successful trajectory requires encountering anywhere from 8–12 messages: *go to the door* (up to 3x), *open the door* (up to 3x), *go to the key*, *pick up the key*, *go to the* [locked] *door*, *open the* [locked] *door*, *go to the ball*, *pick up the ball*.

ObstructedMaze-{1Dl,2Dlhb,1Q}. In these tasks (Figure S9d–f), the agent is also tasked with picking up a ball behind a locked door. In the easier 1Dl task, the key is in the open; in the harder tasks, the keys are hidden in boxes which must be opened, and the doors are blocked by balls which must be moved to access them.

A typical policy trained on each task will encounter anywhere from 66 messages (ObstructedMaze-1Dl) to 244 messages (ObstructedMaze-1Q) throughout training. Of these, a successful trajectory requires anywhere from 6 messages in ObstructedMaze-1Dl (*go to the key*, *pick up the key*, *go to the door*, *open the door*, *go to the ball*, *pick up the ball*) to 11 messages in ObstructedMaze-1Q (*go to the door*, *open the door*, *go to the box*, *open the box*, *pick up the key*, *go to the ball*, *pick up the ball*, *go to the door*, *open the door*, *go to the ball*, *pick up the ball*).

G.2 MiniHack

Language. As discussed previously, it is difficult to enumerate all messages in MiniHack, though we describe the messages encountered for each environment below, with a raw dump in Appendix H. We perform some preprocessing on the messages to prevent unbounded growth: we replace all numbers (e.g. *3 flint stones*, *2 flint stones*, etc) with a single variable *N*; we fix wands encountered in the environment to be a single Wand of Death (instead of having a wand of over 30 different types); items that can be arbitrarily named with random strings (e.g. *a scroll labeled zelgo mer*; *a dog named Hachi* have their names removed; finally, we cap the number of unique messages for each environment at 100.

River. In this task (Figure S9g), the agent must cross the river located to the right on the environment, which can only be done by planning and pushing at least two boulders into the river in a row (to form a bridge over the river). A typical policy will encounter 14 distinct messages during training (Appendix H.1), of which 7–8 messages (3–4 unique) are seen during training (*with great effort you move the boulder*, *you push the boulder into the water.*, *now you can cross it!*; these messages must be repeated twice).

Wand of Death ({Medium,Hard}). In these tasks (Figure S9i–j; also described in the main text), the agent must learn to use a *wand of death*, which can zap and kill enemies. This involves a complex sequence of actions: the agent must find the wand, pick it up, choose to *zap* an item, select the wand in the inventory, and finally choose the direction to zap (towards the minotaur which is pursuing the player). It must then proceed past the minotaur to the goal to receive reward. Taking these actions out of order (e.g. trying to *zap* something with nothing in the inventory, or selecting something other than the wand) has no effect.

In the Medium environment, the agent is placed in a narrow corridor with the wand somewhere in the corridor, and the minotaur is asleep (which gives the agent more time to explore). In the Hard environment, the agent is placed in a larger room where it must first find the wand, with the added challenge that the minotaur is awake and pursues the player, leading to death if the minotaur ever touches the player.

In both Medium and Hard environments, a policy encounters around 60 messages (Appendix H.2), of which 7 messages (7 unique) are typically necessary to complete the task (*you see here a wand*, *f - a wand*, *what do you want to zap?*, *in what direction? you klil the minotaur!*, *welcome to experience level 2*, *you see here a minotaur corpse*).

Quest ({Easy,Medium}). These tasks (Figure S9j–k) require learning to use a *Wand of Cold* to navigate over a river of lava while simultaneously fighting monsters. The agent spawns with a Wand of Cold in its inventory, and must learn to zap the Wand of Cold at the lava, which freezes it and forms a bridge over the lava.

In the Easy environment, the agent must first cross the lava river, then survive fights with one or two monsters to the staircase at the end of the hall. In the Medium environment, the agent must first fight several monsters *before* crossing the lava river. There is an additional challenge: note the narrow corridor before the main room in the Medium environment. If the agent runs into the room and tries to fight the monsters all at once, it quickly will become overwhelmed and die; to successfully kill all monsters, the agent must learn to use the narrow corridor as a "bottleneck", first baiting then leading the monsters into the corridor, until all are defeated. It can then use the Wand of Cold to cross the lava bridge to the staircase beyond.

In both Easy and Medium environments, a typical policy encounters the maximum of 100 messages (Appendix H.3). Very few messages, besides *what do you want to zap? / in what direction ? / the lava cools and solidifies* are *required* to complete an episode, since there is a large variety of monsters faced and uncertainty in their behaviors, which can trigger additional messages. However, highly efficient expert play typically encounters 8–12 messages (7–10 unique) in the Easy environment and 30–40 messages (8–12 unique) in the Medium environment: besides freezing the lava to cross the environment, the rest of the messages are combat related (e.g. *you kill the enemy!*, *an enemy corpse*, additional zapping commands, etc).

MultiRoom-N{2,4}-Extreme. These tasks (Figure S9l–m) are ported from the MiniGrid MultiRoom tasks, but with significant additional challenges. The ultimate task is to reach the last room in a sequence of interconnected rooms, but in the Extreme versions of this task, the walls are replaced with lava (which result in instant death if touched), there are monsters in each room (which must be fought), and additionally the doors are locked and must be "kicked" open (which requires that select the kick action, then kick in the direction of the door).

In both environments, a typical policy encounters the maximum of 100 messages (Appendix H.4). In a successful trajectory, an agent encounters 20–30 (12–14 unique) messages in MultiRoom-N2 and 60–70 messages (16–18 unique) in MultiRoom-N4. These messages are mostly combat-related (*the enemy hits!*, *you hit the enemy!*, *you kill the enemy*, with repeated messages relating to kicking down

F.1. River

```
the stairs are solidly fixed to the floor.
with great effort you move the boulder.
that was close.
it's solid stone.
you don't have anything to zap.
you try to move the border, but in vain.
you try to crawl out of the water.
there is a boulder here, but you cannot lift any more.
there is nothing here to pick up.
never mind.
pheew!
you push the boulder into the water.
now you can cross it!
it sinks without a trace!
```

F.2. WoD-{Medium,Hard}

```
c - a uncursed flint stone ( in quiver pouch ).
c - n uncursed flint stone ( in quiver pouch ).
in what direction?
it's a wall.
you don't have anything to zap.
you see here a wand.
you see here a uncursed flint stone.
you see here n uncursed flint stones.
there is nothing here to pick up.
what a strange direction!
the stairs are solidly fixed to the floor.
there is a staircase up here.
f - a wand.
the death ray bounces!
that is a silly thing to zap.
what do you want to throw?
what do you want to zap?
the wand glows and fades.
the death ray whizzes by you!
nothing happens.
b - a + 2 sling.
the wand turns to dust.
b - a blessed + 2 sling.
the flint stone misses the minotaur.
a wand shatters into a thousand pieces!
c - n uncursed flint stones.
c - a uncursed flint stone.
the flint stone hits another object.
the flint stone hits another object and falls down the stairs.
you see here a minotaur corpse.
welcome to experience level 2.
the flint stone hits the minotaur.
the flint stone hits other objects.
continue? ( q )
from the impact, the other object falls.
movement is very hard.
you stagger under your heavy load.
you kill the minotaur!
the sling misses the minotaur.
you have much trouble lifting a minotaur corpse.
the flint stone falls down the stairs.
the flint stone hits the minotaur!
the death ray misses the minotaur.
you see here a + 2 sling.
from the impact, the other objects fall.
the minotaur butts!
the sling hits the minotaur!
the sling hits the minotaur.
the wand misses the minotaur.
the wand falls down the stairs.
the sling hits another object.
g - a minotaur corpse.
you have extreme difficulty lifting a minotaur corpse.
the wand hits another object.
you see here a blessed + 2 sling.
the minotaur hits!
the flint stone hits other objects and falls down the stairs.
the sling hits another object and falls down the stairs.
the wand hits another object and falls down the stairs.
your movements are now unencumbered.
your movements are slowed slightly because of your load.
you can barely move a handspan with this load!
```

F.3. Quest-{Easy,Medium}

```
a enemy blocks your path.
a crude dagger misses you.
c - a uncursed flint stone ( in quiver pouch ).
f - a horn.
g - a wand.
h - a enemy corpse.
h - a crude dagger.
h - a scroll.
h - n darts.
i - a gem.
the wand glows and fades.
the enemy just misses!
the enemy hits!
the enemy picks up a uncursed flint stone.
the enemy throws a crude dagger!
the enemy touches you!
the enemy bites!
the enemy wields a crude dagger!
the enemy misses!
the enemy thrusts her crude dagger.
the stairs are solidly fixed to the floor.
the bolt of cold bounces!
the flint stone misses the enemy
the lava cools and solidifies.
in what direction?
that is a silly thing to zap.
it's a wall.
you are almost hit by a crude dagger.
you are hit by a crude dagger.
you get zapped!
you stop at the edge of the lava.
you kill the enemy
you miss the enemy
you destroy the enemy
there is nothing here to pick up.
what a strange direction!
what do you want to zap?
your movements are slowed slightly because of your load.
invalid direction for'g'prefix.
h - a wand.
h - n throwing stars.
i - a potion.
the enemy throws a dart!
the enemy escapes the dungeon!
the bolt of cold hits you!
the flint stone hits the enemy
you are hit by a dart.
a lit field surrounds you!
h - a skull cap.
h - a food ration.
h - a towel.
h - a dart.
i - a food ration.
i - a scroll.
i - n darts.
the bolt of cold hits the enemy
the dart misses the enemy
j - a dart.
you are almost hit by a dart.
you cannot escape from the enemy
```

doors (*in what direction?*, *as you kick the door, it crashes open*, *WHAAAAM!*, sometimes repeated up to 3 times per door, of which there are 1–3 doors).

H Raw MiniHack Messages

Messages are located on the next page. Each shows the list of messages encountered by a single agent trained on an environment in the corresponding categories. Separate training runs will encounter different messages.

```
a dart misses you.
h - a potion.
the enemy is killed!
c - n uncursed flint stone ( in quiver pouch ).
i - a wand.
the wand turns to dust.
the crude dagger hits the enemy
the crude dagger slips as the enemy throws it!
you don't have enough stamina to move.
you don't have anything to zap.
you rebalance your load.
nothing happens.
movement is difficult.
i - a conical hat.
you pull free from the enemy
your movements are only slowed slightly by your load.
h - a ring.
h - a egg.
i - a crude dagger.
h - a cursed crude dagger.
i - a ring.
the enemy picks up n crude daggers.
the enemy wields n crude daggers!
the enemy thrusts one of her crude daggers.
the bolt of cold whizzes by you!
the crude dagger welds itself to the goblin's hand!
h - a tripe ration.
you hit the enemy
your movements are now unencumbered.
h - a gem.
i - a enemy corpse.
the enemy picks up a gem.
h - a apple.
the enemy picks up a crude dagger.
h - a clear potion.
i - a skull cap.
you have a little trouble lifting h - a enemy corpse.
you have a little trouble lifting i - a food ration.
the crude dagger misses the enemy
```

F.4. MultiRoom-N{2,4}-Extreme

```
a crude dagger misses you.
a dart misses you.
g - a skull cap.
g - a enemy corpse.
g - a food ration.
h - a enemy corpse.
h - a lock pick.
h - a crude dagger.
whammm!!!
the enemy is killed!
the enemy just misses!
the enemy hits!
the enemy throws a crude dagger!
the enemy touches you!
the enemy jumps, nimbly evading your kick.
the enemy bites!
the enemy wields a crude dagger!
the enemy misses!
the enemy thrusts her crude dagger.
the little dog misses the enemy
the crude dagger misses the enemy
the kitten jumps, nimbly evading your kick.
the kitten bites the enemy
the kitten eats a enemy corpse.
the kitten misses the enemy
in what direction?
as you kick the door, it crashes open!
that hurts!
it burns up!
you are hit by a crude dagger.
you get zapped!
you see no door there.
you hit the enemy
you kill the enemy
you miss the enemy
you cannot escape from the enemy
you pull free from the enemy
you kick the enemy
you kick the kitten.
you kick at empty space.
you destroy the enemy
you strain a muscle.
you swap places with your kitten.
this door is locked.
what a strange direction!
do you want your possessions identified? ( n )
your leg feels better.
never mind.
really attack the little dog? ( n )
really attack the little dog? ( n ) n
really attack the kitten? ( n )
dumb move!
ouch!
g - a potion.
h - a potion.
h - a tripe ration.
the enemy escapes the dungeon!
the enemy misses the little dog.
the crude dagger hits the enemy
the dart misses the enemy
you are almost hit by a dart.
you swap places with your little dog.
do you want to see your attributes? ( n )
thump!
g - n darts.
h - n darts.
i - a enemy corpse.
j - a violet gem.
the enemy blocks your kick.
the enemy strikes at your displaced image and misses you!
the enemy throws a dart!
the little dog bites the enemy
the corpse misses the enemy
you are almost hit by a crude dagger.
you are hit by a dart.
you hear the rumble of distant thunder...
g - a crude dagger.
g - a yellowish gem.
g - n fortune cookies.
h - a wand.
h - n arrows.
the enemy picks up a crude dagger.
the enemy misses sirius.
the crude dagger slips as the enemy throws it!
you stop.
you swap places with hachi.
you swap places with sirius.
hachi misses the enemy
sirius bites the enemy
sirius misses the enemy
h - a skull cap.
sirius is in the way!
g - a scroll.
h - a faded pall.
h - a yellowish gem.
j - a enemy corpse.
the saddled pony drops a enemy corpse.
the saddled pony bites the enemy
g - a ring.
the saddled pony picks up a enemy corpse.
```

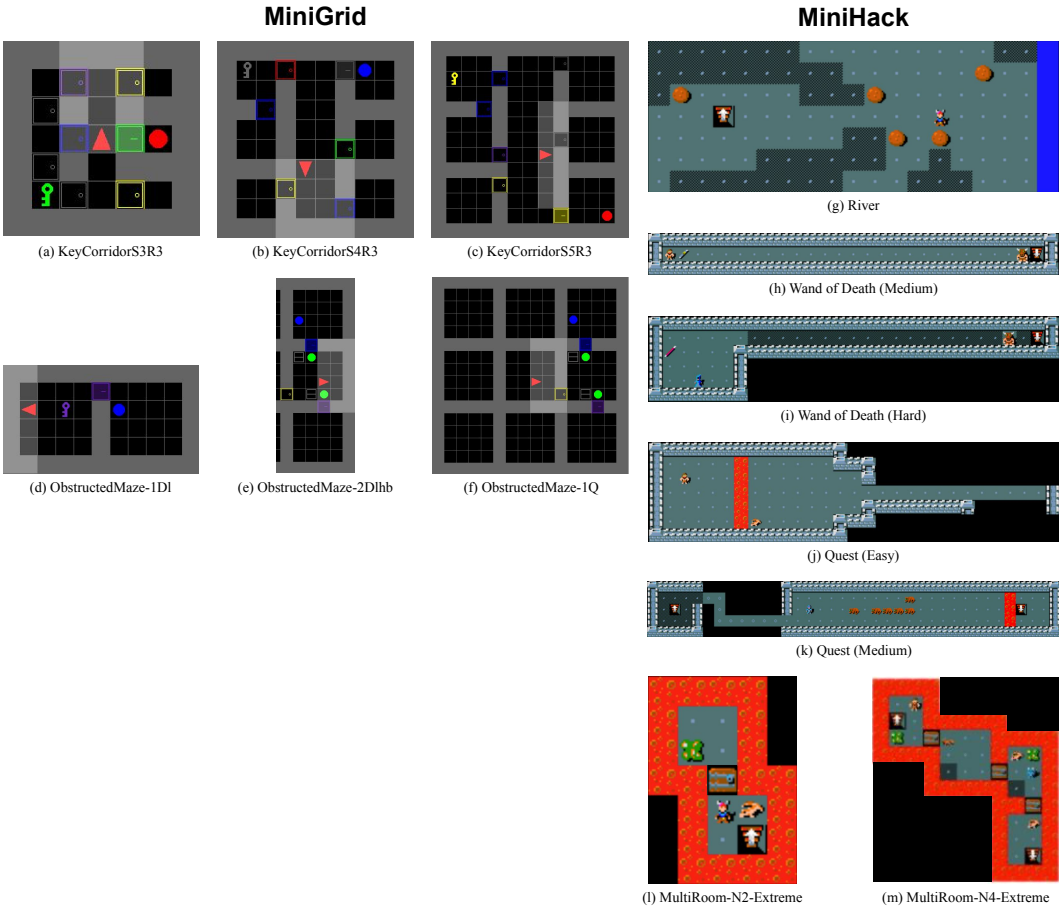

Figure S9: **Examples of all tasks evaluated in this work.** (a–f) MiniGrid tasks; (g–m) MiniHack tasks.