A. Text Manual

Table 5. Example template descriptions. Each underlined word in the example input indicate blanks that may be swapped in the template. Each template takes a word for the object being described (bird, thief, mage), its role (enemy, message, goal) and an adjective (dangerous, secret, crucial).

Example Input
- The bird that is coming near you is the dangerous enemy.
- The secret message is in the thief's hand as he evades you.
- The immovable object is the mage who holds a goal that is crucial.
Enemy Descriptions
Adjectives: dangerous, deadly, lethal
Role: enemy, opponent, adversary
Message Descriptions
Adjectives: restricted, classified, secret
Role: message, memo, report
Goal Descriptions
Adjectives: crucial, vital, essential
Role: goal, target, aim

To collect the text manual, we first crowdsource 82 templates (with 2,214 possible descriptions after filling in the blanks). Each Amazon Mechanical Turk worker is asked to paraphrase a prompt sentence while preserving words in boldface (which become the blanks in our templates). We have three blanks per template, one each for the entity, role and an adjective. For each role (enemy, message, goal) we have three role words and three adjectives that are synonymous (Table 5). Each entity is also described in three synonymous ways. Thus, every entity-role assignment can be described in 27 different ways on the same template. Raw templates are filtered for duplicates, converted to lowercase, and corrected for typos to prevent confusion on downstream collection tasks.

To collect the free form text for a specific entity-role assignment, we first sample a random template and fill each blank with one of the three possible synonyms. The filled template becomes the prompt that is shown to the worker. For each prompt, we obtain two distinct paraphrased sentences to promote response diversity.

On all tasks (template and free-form) we provide an example prompt (which is distinct from the one provided) and example responses to provide additional task clarity. Aside from lower-casing the free-form descriptions and removing duplicate responses, we do no further preprocessing.

To ensure fluency in all responses, we limited workers to those located in the United States with at least 10,000 completed HITs and an acceptance rate of $\geq 99\%$. Some representative responses of free-form responses are presented in table 6. We paid our workers US$0.25 for each pair of sentences, as we found the task was usually finished in $\leq$ 1 min. This translates to a pay of at least $15 per hour per worker.

Table 6. Example descriptions for MESSENGER after the second round of data collection. Note the use of synonyms *flying machine* and *airplane*, which also needs to be disambiguated from *winged creature* (bird). Some descriptions have information divided across two separate sentences. We do not correct typos (*italics*). Some typos (*plan* instead of *plane*) render the description useless, forcing the agent to infer the correct entity form the other descriptions in the text manual.

- the flying machine remains still, and is also the note of upmost secrecy.
- the airplane is coming in your direction. that airplane is the *pivitol* target.
- the winged creature escaping from you is the vital target.
- the fleeing *plan* is a critical target.

B. Environment Details

Table 7. Basic information about our environment MESSENGER. Each game features 3 out of 12 possible unique non-agent entities, with up to 5 non-agent entities total. Each entity is assigned a role of enemy, message or goal.

Entities	bird, dog, fish, scientist, queen, thief, airplane, robot, ship, mage, sword, orb
Roles	enemy, message, goal
Movements	chasing, fleeing, immovable
Total games	$P(12, 3) = 1320$

Details about MESSENGER can be found in table 7. On stage 1 (S1), the three entities start randomly in three out of four possible locations, two cells away from the agent. The agent always begins in the center of the grid. It starts without the message with probability 0.8 and begin with the message otherwise. When the agent obtains the message, we capture this information by changing the agent symbol in the observation.

On stage 2 (S2), the agent and entities are shuffled between four possible starting locations at the start of each episode. On S2, the mobile entities (fleeing, chasing) move at half the speed of the agent. On S2 train, there is always one chasing, one fleeing and one immovable entity. Test games can feature any combination of movement dynamics.

On stage 3 (S3), the agent and non-player entities are shuffled between 6 possible starting locations. As with S2, entities move at half the speed of the agent. The one distractor description may either reference the enemy as a *message* or a *goal*, with a movement type that is distinct from the true movement type of the enemy. S3 test games do not feature unseen movement combinations, since the movements of the entities are integral to the gameplay in S3.

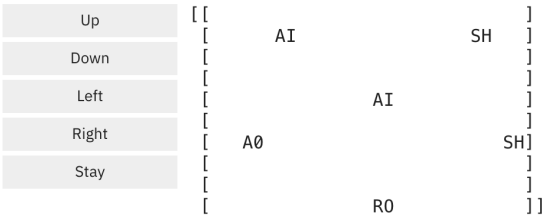

Figure 7. An S3 game on the interface used to collect human playthroughs. `A0` represents the agent and other entities are represented by the first two letters of the entity name in Table 7.

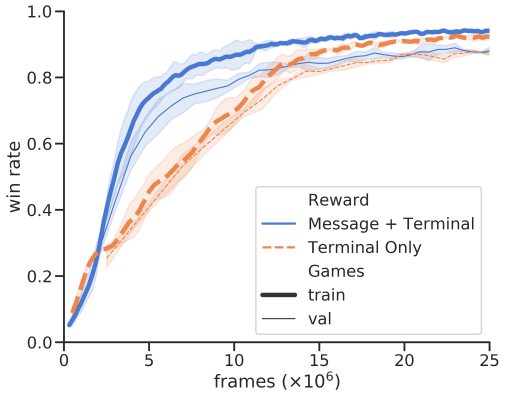

Figure 8. Win rates of EMMA on MESSENGER with intermediate rewards (Message + Terminal) and terminal rewards only (Terminal Only) on S2 games. Results are over three seeds and shaded area indicates standard deviation.

Since there are only 4 single-combination (SC) training games and 40 multi-combination (MC) training games, we sample the games non-uniformly at the start of each episode to ensure that there is enough interaction with SC entities to induce an entity grounding. On all stages we sample an SC game with probability 0.25 and an MC game otherwise. Not all descriptions have movement type information (e.g. 'chasing'). We also collect *unknown type* descriptions with no movement type information. During training, in S1 and S2, each description is independently an unknown type description with probability 0.15. On S3, we do not provide any description with no movement information, since this would render disambiguation via movement differences impossible.

Human Playthroughs We collect expert human playthroughs using the interface presented in Figure 7. The human expert has access to the manual, navigation commands, and a text-rendered grid observation. The grid observation uses the first two letters of the entity name from Table 7 to represent each entity. Thus, human performance does not reflect the challenge of grounding entities by playing the environment; rather it quantifies the difficulty of completing the task with entity groundings provided upfront.

Terminal Rewards On S2, we provide an intermediate scalar reward of 0.5 for obtaining the message. To assess whether only terminal rewards is sufficient for EMMA to learn a good policy on MESSENGER, we evaluate EMMA on S2 using ± 1 terminal rewards in Figure 8. Intermediate rewards help EMMA converge to a higher win rate slightly faster, but EMMA can converge to the same win rate using just terminal rewards.

Negation We procedurally generate the negated text by negating existential words (e.g. 'is an enemy' becomes 'is not an enemy'). We manually negate those descriptions not captured by the rules. During both training and evaluation, we provide a complete text manual without any negated description with 0.75 probability, and randomly select a description in the manual to negate otherwise. When we negate an entity description z_e to z'_e, we also change the role ('...is an enemy' becomes '...is not a goal', for example). Thus the information present in the manual has not changed, but the agent must look at the remaining two descriptions to deduce the role of e with description z'_e.

Transfer Learning We test transfer by introducing two new entities – a trap and a gold which provide rewards of -1 and 1 respectively. Both collectables are randomly shuffled between two possible starting locations at the start of each episode and do not move. We train the models in this new setting in a multi-task fashion on the 32 validation games. After the agent encounters either the trap or gold, the collected item disappears. Neither item terminates the episode and the agent can still win or lose the current episode regardless of whether it has picked up the gold or trap.

B.1. Comparison with RTFM

The main novelty of our work (both the MESSENGER environment and our model) is in specifically tackling the issue of *entity grounding* without any prior knowledge. To do this, MESSENGER in contrast to RTFM (1) does not have any signal connecting entities to text references, (2) features much richer language, and (3) requires interaction in the environment to ground entities to text. We describe these in more detail:

1. RTFM's observation space consists of a grid of text in which entity names are identical to their correspond-

ing references in the manual. Thus, both the text in the manual and the observation are embedded into the same space (e.g. using the same word vectors), essentially providing models with the entity grounding upfront. In contrast, our environment has a separate set of symbols for the entities with no relation to the text in our manual. Thus, the entities and text are embedded into different spaces, and learning to map between these two spaces is the key challenge in our environment that has not been explored before.

2. RTFM features only 32 total rule-based templates for the text, and each entity can only be referred to in a single way (`goblin` is always 'goblin'). In contrast, we crowdsourced thousands of completely free-form descriptions in *two* rounds using Amazon Mechanical Turk. After obtaining the seed templates from the first round, we intentionally inject multiple synonyms for each entity to construct each prompt for the second round. Workers often further paraphrased these synonyms, resulting in 5, 6 or often more ways to describe the same entity (e.g. 'airplane', 'jet', 'flying machine', 'aircraft', 'airliner' all describe `plane`.). The need to map these different text references to the same entity symbol further complicates the entity grounding problem in our case and more closely mirrors the challenges of grounding entities in the real world. We believe MESSENGER provides a much closer approximation to natural language compared to RTFM.

3. RTFM features all possible combinations of entities during training which provides an additional signal that may simplify the grounding problem.

4. Each entity in RTFM only moves in a single way, whereas in MESSENGER, each entity may have different dynamics such as fleeing, chasing, and immovable entities (and this is also described in the text). This also allows us to test our model's ability to generalize to unseen dynamics with unseen entity movement combinations, whereas in RTFM the evaluation on unseen games is essentially state-estimation.

MESSENGER shares many aspects with RTFM (e.g. grid-world with different entities and goals). That said, there are numerous reasons why we were not able to adapt the original RTFM environment to meet our requirements. We enumerate them here:

1. The dynamics in RTFM make entity grounding (the primary focus of our work) difficult. MESSENGER requires much simpler reasoning than RTFM, and it is already too difficult to ground entities directly in MESSENGER without a curriculum. RTFM sidesteps the issue by providing this grounding beforehand.

2. Obtaining enough crowdsourced descriptions is hard with RTFM because of the more complicated dynamics. In RTFM, there are monsters, weapons, elements, modifiers, teams, variable goals and different weaknesses between entity types that need to be specified. Collecting enough descriptions that are entirely human written would be challenging. (RTFM sidesteps this issue by using templates to generate their text manual). In contrast, there are only entities, 3 roles, and a fixed goal in MESSENGER, making the text-collection task much more tractable.

3. The entities in our MESSENGER environment are carefully chosen to make entity grounding harder. In RTFM, each entity is referred to in a single way, and it is not clear how to refer to them in multiple ways (e.g. there are not too many other ways to say 'goblin'). In contrast, we specifically chose a set of entities that allowed for multiple ways of description, and actively encouraged this during data collection.

4. The combination of entities that appear during training in MESSENGER is carefully designed. This is so that we can introduce single-combination games and the associated grounding challenges that come with it.

5. We have different movement types for each entity. These different movements are referred to in our text manual and significantly increase the richness and variety of descriptions we collected, and also allow us to test generalization to unseen movement combinations. In RTFM, the entity movements are the same and fixed for all entities.

6. Each entity's attribute is referenced in the observation in RTFM, e.g. the grid has entries such as `fire goblin`. We could add to the cell an extra symbol for `fire`, but this further obfuscates the entity grounding problem we are focusing on, because we would also need to obtain a grounding for all the attributes such as `fire`.

C. Implementation and Training Details

All models are end-to-end differentiable and we train them using proximal policy optimization (PPO) (Schulman et al., 2017) and the Adam optimizer (Kingma & Ba, 2015) with a constant learning rate of 5×10^{-5}. We also evaluated learning rates of 5×10^{-4} and unroll lengths of 32 and 64 steps by testing on the validation games. On S1, S2 and S3 we limit each episode to 4, 64, and 128 steps respectively and provide a reward of -1 if the agent does not complete the objective within this limit. Note that the computation of random agent performance is also subject to these step constraints.

For all experiments we use $d = 256$. When multiple entities E' overlap in the observation, we fill the overlapping cell with the average of the entity representations $\frac{1}{|E'|}\sum_{e\in E'} x_e$. The convolutional layer consists of 2×2 kernels with stride 1 and 64 feature maps. The FFN in the action module is fully-connected with 3 layers and width of 128. To give the Mean-BOS and G-ID baselines (Fig. 9) the ability to handle the additional conditioning information, we introduce an additional layer of width 512 at the front of the FFN for those baselines only. Between each layer, we use leaky ReLU as the activation function.

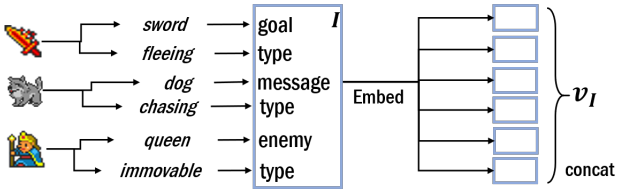

Figure 9. G-ID model

We pretrain BAM on 1.5×10^6 episodes. If two descriptions map to the same entity, we take the one with higher $P(e|z)$, and If an entity receives no assignment we represent it with a learned default embedding $\mathrm{Emb}(e)$. txt2π is trained using 10-12 actors, a model dimension of 128, and a learning rate of 0.0002.

We train models for up to 12 hours on S1, 48 hours on S2 and 72 hours on each S3. We use the validation games to save the model parameters with the highest validation win rate during training and use these parameters to evaluate the models on the test games. All experiments were conducted on a single Nvidia RTX2080 GPU.

D. Model Design

The weights u_k and u_v were introduced to make sure that the token embeddings for filler words such as 'the', 'and', 'or' do not drown out the words relevant to the task when we take the average in equations 1 and 2. Qualitatively, we observe that u_k learns to focus on tokens informative for identifying the entity (e.g. `mage`, `sword`) while u_v learns to focus on tokens that help identify the entities' roles (e.g. `enemy`, `message`).

We also found that using a pretrained language model was critical for success due to the large number of ways to refer to a single entity (e.g. 'airplane', 'jet', 'flying machine', 'aircraft', 'airliner' all refer to `plane`).

D.1. Model Variations

We consider a variation to EMMA. Instead of obtaining token weights α, β in equations 1 and 2 by taking a softmax over the token-embedding and vector products $u_k \cdot t$ and $u_v \cdot t$, we consider independently scaling each token using a sigmoid function. Specifically, we will obtain key and value vectors k_z and v_z using:

$$k_z = \sum_{i=1}^{n} \frac{S(u_k \cdot t_i)}{\sum_{i=1}^{n} S(u_k \cdot t_i)} W_k t_i + b_k \tag{9}$$

$$v_z = \sum_{i=1}^{n} \frac{S(u_v \cdot t_i)}{\sum_{i=1}^{n} S(u_v \cdot t_i)} W_v t_i + b_v \tag{10}$$

where S is the logistic sigmoid function, and all other details are identical to EMMA. We call this model EMMA-S. We notice that both EMMA and EMMA-S are able to obtain good training and validation performance, whith EMMA-S obtaining higher rewards on S2. However, on S1, EMMA is able to obtain a higher validation reward faster (Fig. 10). Moreover, EMMA can learn robust groundings even with neutral entities, while EMMA-S often overfits to a spurious grounding with neutral entities (Fig. 11). Although the independent scaling in EMMA-S allows the model to consider more tokens simultaneously, the softmax selection of EMMA facilitates more focused selection of relevant tokens, and this may help prevent overfitting.

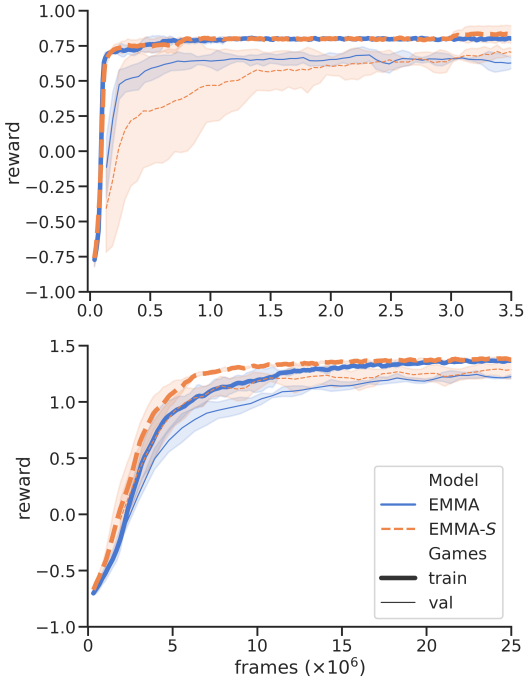

Figure 10. Average episodic rewards on S1 **(top)** and S2 **(bottom)** on training (thick line) and validation (thin line) games, as a function of training steps (x-axis) for both EMMA (solid line) and EMMA-S (dotted line). Both models are able to perform well, however, EMMA is able to obtain a good validation reward faster. All results are averaged over three seeds and shaded area indicates standard deviation.

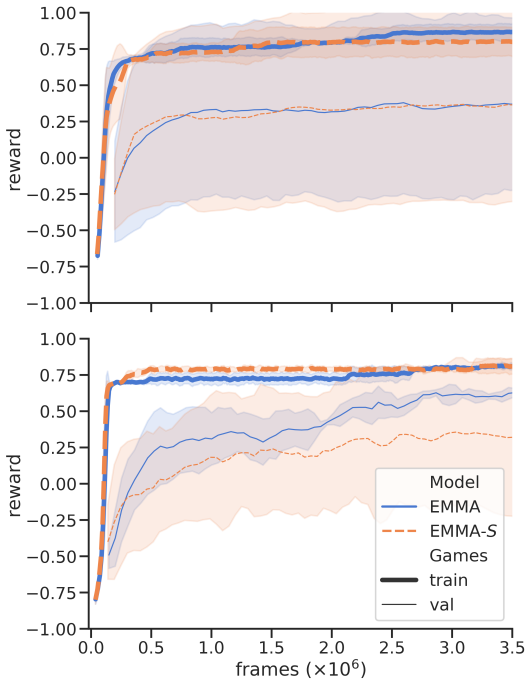

Figure 11. Average episodic rewards on S1 games with negation **(top)** and neutral entities **(bottom)** on training (thick line) and validation (thin line) games, as a function of training steps (x-axis) for both EMMA (solid line) and EMMA-S (dotted line). Both models struggle on negation, but EMMA is able to perform well with neutral entities. All results are averaged over three seeds and shaded area indicates standard deviation. Note the shared x-axis.

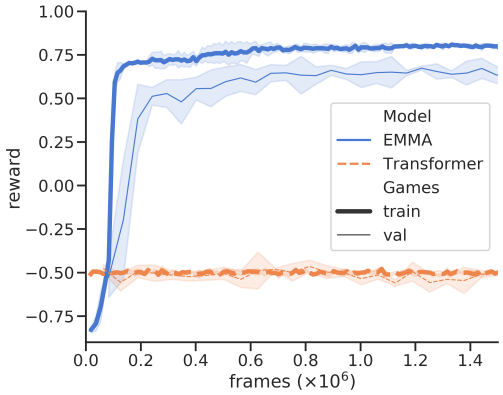

Figure 12. Average episodic rewards on S1 games with as a function of training steps (x-axis) for both EMMA (solid line) and a baseline agent consisting of a BERT model that ingests the manual and state observation converted to a string (dotted line). While EMMA is able to fit to both training and validation games, the transformer baseline struggles to learn. All results are averaged over three seeds and shaded area indicates standard deviation.

D.2. Comparison with Transformer

EMMA relies heavily on the dot-product attention mechanism to extract relevant information from the text manual. To assess the extent that attention alone is sufficient for solving MESSENGER, we train a Transformer (Vaswani et al., 2017) on MESSENGER.

Specifically, we use a pretrained BERT-base model (Devlin et al., 2019) that is identical to the one used by EMMA. We first concatenate the text descriptions $d_1, ..., d_n$ to form the manual string s_m. For each entity in the observation, we generate a string s_e by indicating the x and y coordinates for every entity e as follows: 'e: x, y;'. We then convert the entire grid observation into a string s_o by concatenating s_e for every entity e in the observation. The final input to BERT is then s_m `[SEP]` s_o. We train action and value MLPs on top of the `[CLS]` representation in the final layer of the BERT model. The MLPs are identical to the ones used in EMMA. The entire model is end-to-end differentiable and we train it using PPO using an identical setup to the one used to train EMMA.

The results of training this Transformer baseline on S1 is presented in Figure 12. While EMMA is able to fit to both training and validation games, the rewards for the Transformer baseline do not significantly increase even after 1.5×10^6 steps. We hypothesize that the difficulty of encoding spatial information in text form makes it very difficult for this model to learn a performant policy on MESSENGER.