Collaborative Dialogue in Minecraft

Anjali Narayan-Chen* **Prashant Jayannavar*** **Julia Hockenmaier**
University of Illinois at Urbana-Champaign
{nrynchn2, paj3, juliahmr}@illinois.edu

Abstract

We wish to develop interactive agents that can communicate with humans to collaboratively solve tasks in grounded scenarios. Since computer games allow us to simulate such tasks without the need for physical robots, we define a Minecraft-based collaborative building task in which one player (**A**, the Architect) is shown a target structure and needs to instruct the other player (**B**, the Builder) to build this structure. Both players interact via a chat interface. **A** can observe **B** but cannot place blocks. We present the Minecraft Dialogue Corpus, a collection of 509 conversations and game logs. As a first step towards our goal of developing fully interactive agents for this task, we consider the subtask of Architect utterance generation, and show how challenging it is.

1 Introduction

Building interactive agents that can successfully communicate with humans about the physical world around them to collaboratively solve tasks in this environment is a long-sought goal of AI (e.g. Winograd, 1971). Such situated dialogue poses challenges that go beyond what is required for the slot-value filling tasks performed by standard dialogue systems (e.g. Kim et al., 2016, 2017; Budzianowski et al., 2018) or chatbots (e.g. Ritter et al., 2010; Schrading et al., 2015; Lowe et al., 2015), as well as for so-called visual dialogue where users talk about a static image (Das et al., 2017) or video-context dialogue where users interact in a chat room while viewing a live-streamed video (Pasunuru and Bansal, 2018). It requires the ability to refer to real-world objects and spatial relations that depend on the current position of the speakers as well as changes in the environment. Due to the expense of actual human-robot communication (e.g. Tellex et al., 2011; Thomason et al., 2015; Misra et al., 2016; Chai et al., 2018), simulated environments that allow easier experimentation are commonly used (Koller et al., 2010; Chen and Mooney, 2011; Janarthanam et al., 2012).

In this paper, we therefore introduce the Minecraft Collaborative Building Task, in which pairs of users control avatars in the Minecraft virtual environment and collaboratively build 3D structures in a Blocks World-like scenario while communicating solely via text chat (Section 3). We have built a data collection platform and have used it to collect the Minecraft Dialogue Corpus, consisting of 509 human-human written dialogues, screenshots and complete game logs for this task (Section 4). While our ultimate goal is to develop fully interactive agents that can collaborate with humans successfully on this task, we first consider the subtask of Architect utterance generation (Section 5) and describe a set of baseline models that encode both the dialogue history (Section 6) and the world state (Section 7). Section 8 describes our experiments. Our analysis (Section 9) highlights the challenges of this task. The corpus and platform as well as our models are available for download. [1]

2 Related Work

Our work is partly inspired by the HCRC Map Task Corpus (Anderson et al., 1991), which consists of route-following dialogues between an Instruction Giver and a Follower who are given maps of an environment that differ in significant details. Our task also features asymmetric roles and levels of information between the two speakers, but operates in 3D space and focuses on the creation of structures rather than navigation around existing ones. Koller et al. (2010) design a challenge where systems with access to symbolic world rep-

*Both authors equally contributed to the paper.

[1] http://juliahmr.cs.illinois.edu/Minecraft

resentations and a route planner generate real-time instructions to guide users through a treasure hunt in a virtual 3D world.

There is a resurgence of interest in Blocks World-like scenarios. Wang et al. (2017) let users define 3D voxel structures via a highly programmatic natural language. The interface learns to understand descriptions of increasing complexity, but does not engage in a back-and-forth dialogue with the user. Most closely related to our work are the corpora of Bisk et al. (2018, 2016a,b), which feature pairs of scenes involving simulated, uniquely labeled, 3D blocks annotated with single-shot instructions aimed at guiding an (imaginary) partner on how to transform an input scene into the target. In their scenario, the building area is always viewed from a fixed bird's-eye perspective. Simpler versions of the data retain the grid-based assumption over blocks, and structures consist solely of numeric digits procedurally reconstructed along the horizontal plane. Later versions increase the task complexity significantly by incorporating human-generated, truly 3D structures and removed the grid assumption, as well as allowing for rotations of individual blocks. Their blocks behave like physical blocks, disallowing structures with floating blocks that are prevalent in our data. Our work differs considerably in a few other aspects: our corpus features two-way dialogue between an instructor and a real human partner; it also includes a wide range of perspectives as a result of using Minecraft avatars, rather than a fixed bird's-eye perspective; and we utilize blocks of different colors, allowing for entire substructures to be identified (e.g., "the red pillar").

3 Minecraft Collaborative Building Task

Minecraft (`https://minecraft.net/`) is a popular multi-player game in which players control avatars to navigate in a 3D world and manipulate inherently block-like materials in order to build structures. Players can freely move, jump and fly, and they can choose between first- or third-person perspectives. Camera angles can be smoothly rotated by moving around or turning one's avatar's head up, down, and side-to-side, resulting in a wide range of possible viewpoints.

Blocks World in Minecraft Minecraft provides an ideal setting to simulate Blocks World, although there are two key differences to physical toy blocks: Minecraft blocks can only be placed on a discrete 3D grid, and they do not need to obey gravity. That is, they do not need to be placed on the ground or on top of another block, but can be put anywhere as long as one of their sides touches another block. That neighboring block can later be removed, allowing the second block (and any structure supported by it) to "float". Players need to identify when such supporting blocks need to be added or removed.

Collaborative Building Task We define the Collaborative Building Task as a two-player game between an Architect (**A**) and a Builder (**B**). **A** is given a target structure (*Target*) and has to instruct **B** via a text chat interface to build a copy of *Target* on a given build region. **A** and **B** can communicate back and forth via chat throughout the game (e.g. to resolve confusions or to correct **B**'s mistakes). **B** is given access to an inventory of 120 blocks of six given colors that it can place and remove. **A** can observe **B** and move around in its world, allowing it to provide instructions from varying perspectives. But **A** cannot move blocks, and remains invisible to **B**. The task is complete when the structure built by **B** (*Built*) matches *Target*, invariant to translations within the horizontal plane and rotations about the vertical axis. *Built* also needs to lie completely within the boundaries of the predefined build region.

Although human players were able to complete each structure successfully, this task is not trivial. Figure 1 shows the perspectives seen by each player in the Minecraft client. This example from our corpus shows some of the challenges of this task. **A** often provides instructions that they think are sufficient, but leave **B** still clearly confused, indicated either by **B**'s lack of initiative to start building or a confused response. Once a multi-step instruction is understood, **B** also needs to plan a sequence of steps to follow that instruction; in many cases, **B** chooses clearly suboptimal solutions, resulting in large amounts of redundancy in block movements. A misinterpreted instruction may also lead to a whole sequence of blocks being misplaced by **B** (either due to miscommunication, or because **B** made an educated guess on how to proceed) until **A** decides to intervene (in the example, this can be seen with the built yellow 6). **A** could also misinterpret the target structure, giving **B** incorrect instructions that would later need to be rectified. This illustrates the challenges involved

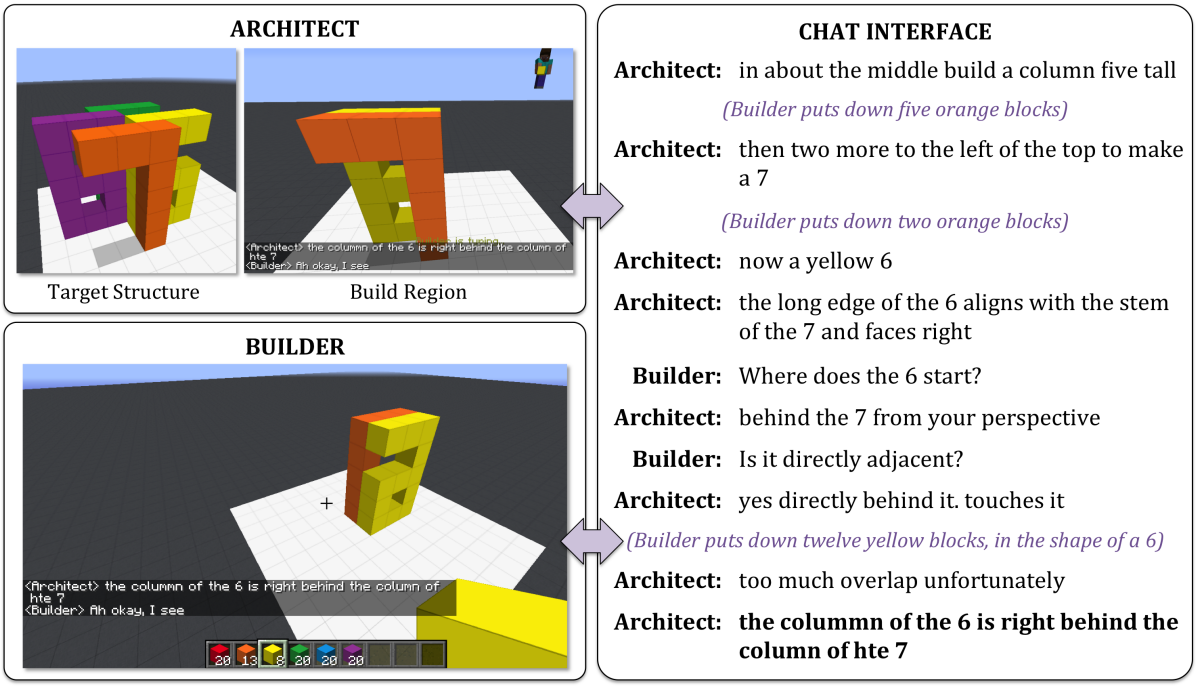

Figure 1: In the Minecraft Collaborative Building Task, the Architect (**A**) has to instruct a Builder (**B**) to build a target structure. **A** can observe **B**, but remains invisible to **B**. Both players communicate via a chat interface. (NB: We show **B**'s actions in the dialogue as a visual aid to the reader.)

in designing an interactive agent for this task: the Architect needs to provide clear instructions; the Builder needs to identify when more information is required, and both agents may need to design efficient plans to construct complex structures.

4 The Minecraft Dialogue Corpus

The Minecraft Dialogue Corpus consists of 509 human-human dialogues and game logs for the Collaborative Building Task. This section describes this corpus and our data collection process. Further details are in the supplementary materials.

4.1 Data Collection Procedure

Data was collected over the course of 3 weeks (approx. 62 hours overall). 40 volunteers, both undergraduate and graduate students with varying levels of proficiency with Minecraft, participated in 1.5 hour sessions in which they were paired up and asked to build various predefined structures within a $11 \times 11 \times 9$ sized build region. Builders began with an inventory of 6 colors of blocks and 20 blocks of each color. After a brief warm-up round to become familiar with the interface, participants were asked to successfully build as many structures as they could manage within this time frame. On average, each game took 8.55 minutes.

Architects were encouraged not to overwhelm the Builder with instructions and to allow their partner a chance to respond or act before moving on. Builders were instructed not to place blocks outside the specified build region and to stay as faithful as possible to the Architect's instructions. Both players were asked to communicate as naturally as possible while avoiding idle chit-chat.

Participants were allowed to complete multiple sessions if desired; we ensured that an individual never saw the same target structure twice, and attempted as much as possible to pair them with a previously unseen partner. While some individuals indicated a preference towards either the Architect or Builder roles, roles were, for the most part, assigned in such a way that each individual who participated in repeat sessions played both roles equally often. Each participant is assigned a unique anonymous ID across sessions.

4.2 Data Structures and Collection Platform

Microsoft's Project Malmo (Johnson et al., 2016) is an AI research platform that provides an API for Minecraft agents and the ability to log, save, and load game states. We have extended Malmo into a data collection platform. We represent the progression of each game (involving the construction of a single target structure by an Architect and

Builder pair) as a discrete sequence of game states. Although Malmo continuously monitors the game, we selectively discretize this data by only saving snapshots, or "observations," of the game state at certain triggering moments (whenever **B** picks up or puts down a block or when either player sends a chat message). This allows us to reduce the amount of (redundant) data to be logged while preserving significant game state changes. Each observation is a JSON object that contains the following information: 1) a time stamp, 2) the chat history up until that point in time, 3) **B**'s position (a tuple of real-valued x, y, z coordinates as well as pitch and yaw angles, representing the orientation of their camera), 4) **B**'s block inventory, 5) the locations of the blocks in the build region, 6) screenshots taken from **A**'s and **B**'s perspectives. Whenever **B** manipulates a block, we also capture screenshots from four invisible "Fixed Viewer" clients hovering around the build region at fixed angles.

4.3 Data Statistics and Analysis

Overall statistics The Minecraft Dialogue Corpus contains 509 human-human dialogues (15,926 utterances, 113,116 tokens) and game logs for 150 target structures of varying complexity (min. 6 blocks, max. 68 blocks, avg. 23.5 blocks). We collected a minimum of three dialogues per structure. The training, test and development sets consist of 85 structures (281 dialogues), 39 structures (137 dialogues), and 29 structures (101 dialogues) respectively. Dialogues for the same structure are fully contained within a single split; structures in training are thus guaranteed to be unseen in test.

On average, dialogues contain 30.7 utterances: 22.5 Architect utterances (avg. length 7.9 tokens), 8.2 Builder utterances (avg. length 2.9 tokens), and 49.5 Builder block movements. Dialogue length varies greatly with the complexity of the target structure (not just the number of blocks, but whether it requires floating blocks or contains recognizable substructures).

Floating blocks Blocks in Minecraft can be placed anywhere as long as they touch an existing block (or the ground). If such a supporting block is later removed, the remaining block (and any structure supported by it) will continue to "float" in place. This makes it possible to produce complex designs. 53.6% of our target structures contain such floating blocks. Instructions for these structures varied greatly, ranging from step-by-step instructions involving temporary supporting blocks to single-shot descriptions such as, simply, "build a floating yellow block" (sufficient for a veteran Minecraft player, but not necessarily for a novice).

Referring expressions and ellipsis Architects made frequent use of implicit arguments and references, relying heavily on the Builder's current perspective and their most recent actions for reference resolution. For instance, Architect instructions could include references such as *"two more in the same direction," "one up," "two towards you,"* and *"one right from the last thing you built."*

Recognizable shapes and sub-structures Some target structures were designed with commonplace objects in mind. Some Architects took advantage of this in their instructions, ranging from straightforward (*'L'*-shapes, *"staircases"*) to more eccentric descriptions (*"either a chicken or a gun turret," "a heart that looks diseased," "a silly multicolored worm"*). To avoid slogging through block-by-block instructions, Architects frequently used such names to refer to sub-elements of the target structure. Some even defined new terms that get re-used across utterances: **A**: *i will refer to this shape as* **r-windows** *from here on out...* **B**: *okay* **A**: *please place the first green block in the right open space of the blue* **r-window**.

Builder utterances Even though the Architect shouldered the large responsibility of describing the unseen structure, the Builder played an active role in continuing and clarifying the dialogue, especially for more complex structures. Builders regularly took initiative during the course of a dialogue in a variety of ways, including verification questions (*"is this ok?"*), clarification questions (*"is it flat?"* or *"did I clean it up correctly?"*), status updates (*"i'm out of red blocks"*), suggestions (*"feel free to give more than one direction at a time if you're comfortable," "i'll stay in a fixed position so it's easier to give me directions with respect to what i'm looking at"*), or extrapolation (*"I think I know what you want. Let me try,"* then continuing to build without explicit instruction).

5 Architect Utterance Generation Task

Although the Minecraft Dialogue Corpus was motivated by our ultimate goal of building agents that can successfully play an entire collaborative building game as Architect or Builder, we first con-

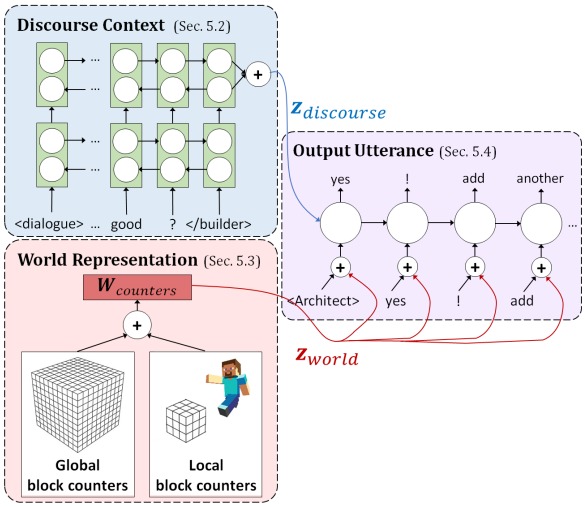

Figure 2: An overview of the full model combining global and local world representation variants.

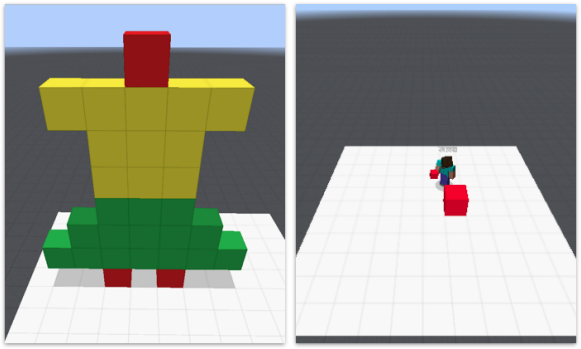

Figure 3: A target structure (left) and corresponding built structure at a certain point in the game (right).

sider the task of Architect utterance generation: given access to the entire game state context leading up to a certain point in a human-human game at which the human Architect spoke next, we aim to generate a suitable Architect utterance.

Architect utterance generation is a much simpler task than developing a fully interactive Architect or Builder, but it still captures some of the essential difficulties of the Architect's role. Since Architects need to be able to give instructions, correct Builders' mistakes and answer their questions, they need the ability to compare the built structure against the target structure, and to understand the preceding dialogue. We also believe that the models developed for this task could be leveraged to at least bootstrap a fully interactive Architect (which will also need to decide when to speak, as well as deal with potentially much noisier dialogue histories than those we are considering here).

Although future work should consider the task of Builder utterance generation, the challenges in creating a fully interactive Builder lie more in the need to understand and execute complex instructions in a discourse and game context, to know when it is appropriate to ask clarification questions and to understand the Architect's answers, than in the need to generate complex utterances.

6 Seq2Seq Architect Utterance Model

We define a sequence of models for Architect utterance generation. Our most basic variant is a sequence-to-sequence model (Sutskever et al., 2014) that conditions the next utterance on the preceding dialogue. Since Architects need to compare the current state of the build region against the target structure, we augment this model in the next section with world state information.

Dialogue History Encoder We encode the entire dialogue history as a sequence of tokens in which each player's utterances are contained within speaker-specific start and end tokens (`<A>...</A>` or `<B>...</B>...`). Each utterance corresponds to a single chat message, and may consist of multiple sentences. These tokens are fed through a word embedding layer and subsequently passed through a bidirectional RNN (Schuster and Paliwal, 1997) to produce an embedding of the entire dialogue history in the encoder RNN's final hidden state.

Output Utterance Decoder The output utterance is generated by a decoder RNN conditioned on the discourse context. In standard fashion, the final hidden state of the encoder RNN is used to initialize the hidden state of the decoder RNN.

7 World State Representations

To be able to give accurate instructions, the Architect requires a mental model of how the target structure can be constructed successfully given the current state of the built structure. Since the Builder's world is not explicitly aligned to the target structure (our space does not contain any markers that would indicate cardinal directions or other landmarks, and we consider any built structure a success as long as it matches the target structure and fits completely into the Builder's build region), this model must consider all possible translational and rotational alignment variants, although we assume it can ignore any sub-optimal alignments. For any given alignment, we compute

the Hamming distance between the built structure and the target (the total number of blocks of each color to be placed and removed), and only retain those alignments that have the smallest distance to the target. Once the game has progressed sufficiently far, there is often only one optimal alignment between built and target structures, but in the early stages, a number of different optimal alignments may be possible. Our world state representation captures this uncertainty.

Figure 3 depicts a target structure (left) and a point in the game at which a single red block has been placed (right). We can identify three potential paths (left, up, and down) to continue the structure by extending it along the four cardinal directions. A permissibility check disqualifies the option of extending to the right, as blocks would end up placed outside the build region. These remaining paths, considered equally likely, indicate the colors and locations of blocks to be placed (or removed). A summary of this information forms the basis of the input to our model.

Computing the distance between structures Computing the Hamming distance between the built and target structure under a given alignment tells us also which blocks need to be placed or removed. A structure S is a set of blocks (c, x, y, z). Each block has a color c and occupies a location (x, y, z) in absolute coordinate space (i.e., the coordinate system defined by the Minecraft client). A structure's position and orientation can be mutated by an alignment A in which S undergoes a translation A_T (shift) followed by a rotation A_R, denoted $A(S) = A_R(A_T(S))$. We only consider rotations about the vertical axis in 90-degree intervals, but allow all possible translations along the horizontal plane. The symmetric difference between the target T and a built structure S w.r.t. an alignment A, $\text{diff}(T, S, A)$, consists of the set of blocks to be placed, $B_p = A(T) - S$ and the set of blocks to be removed from S, $B_r = S - A(T)$.

$$\text{diff}(T, S, A) = B_p \cup B_r$$

The cardinality $|\text{diff}(T, S, A)|$ is the Hamming distance between $A(T)$ and S.

Feasible next placements Architects' instructions often concern the immediate next blocks to be placed. Since new blocks can only be feasibly placed if one of their faces touches the ground or another block, we also wish to capture which blocks B_n can be placed in the immediate next action. B_n, the set of blocks that can be feasibly placed, is a subset of B_p.

Block counters To obtain a summary representation of the optimal alignments (without detailed spatial information), we represent each of the sets B_p and B_r (as well as B_n) of an alignment $A = B_p \cup B_r$ as sets of counters over block colors (indicating how many blocks of each color remain to be placed [next] and to be removed). We compute the set of expected block counters for each color $c \in \{\text{red}, \text{blue}, \text{orange}, \text{purple}, \text{yellow}, \text{green}\}$ and action $a \in \{p, r, n\}$ as the average over all k optimal alignments $A^* = \arg\min_A(|\text{diff}(T, S, A)|)$.

$$E[\text{count}_{c,a}] = \frac{1}{k} \sum_{i=1}^{k} \text{count}^i_{c,a}$$

With six colors, and three sets of blocks (all placements, next placements, removals), we obtain an 18-dimensional vector of expected block counts.

7.1 Block Counter Models

We augment our basic seq2seq model with two variants of block counters that capture the current state of the built structure:

Global block counters are 18-dimensional vectors (capturing expected overall placements, next placements, and removals for each of the six colors) that are computed over the whole build region.

Local block counters Since many Builder actions involve locations immediately adjacent to their last action, we construct local block counters that focus on and encode spatial information of this concentrated region. Here, we consider a $3 \times 3 \times 3$ cube of block locations: those directly surrounding the location of the last Builder action as well as the last action itself. We compute a separate set of block counters for each of these 27 locations. Using the Builder's position and gaze, we deterministically assign a relative direction for each location that indicates its position relative to the last action in the Builder's perspective, e.g., "left", "top", "back-right," etc. The 27 18-dimensional block counters of each location are concatenated, using a fixed canonical ordering of the assigned directions.

Adding block counters to the model To add block counters to out models, we found the best results by feeding the concatenated global and local

counter vectors through a single fully-connected layer before concatenating them to the word embedding vector that is fed into the decoder at each time step (Figure 2).

8 Experimental Setup

Data Our training, test and dev splits contain 6,548, 2,855, and 2,251 Architect utterances.

Training We trained for a maximum of 40 epochs using the Adam optimizer (Kingma and Ba, 2015). During training, we minimize the sum of the cross entropy losses between each predicted and ground truth token. We stop training early when perplexity on the held-out validation set had increased monotonically for two epochs. All word embeddings were initialized with pretrained GloVe vectors (Pennington et al., 2014). We first performed grid search over model architecture hyperparameters (embedding layer sizes and RNN layer depths). Once the best-performing architecture was found, we then varied dropout parameters (Srivastava et al., 2014). More details can be found in the supplementary materials.

Decoding We use beam search decoding to generate the utterance with the maximum log-likelihood score according to our model normalized by utterance length (beam size $= 10$). In order to promote diversity of generated utterances, we use a γ penalty (Li et al., 2016) of $\gamma = 0.8$. These parameters were found by a grid search on the validation set for our best model.

9 Results and Analysis

We evaluate our models in three ways: we use automated metrics to assess how closely the generated utterances match the human utterances. For a random sample of 100 utterances per model, we use human evaluators to identify dialogue acts and to evaluate whether the generated utterances are correct in the given game context. Finally, we perform a qualitative analysis of our best model.

9.1 Automated Evaluation

Metrics To evaluate how closely the generated utterances resemble the human utterances, we report standard BLEU scores (Papineni et al., 2002). We also compute (modified) precision and recall of a number of lists of domain-specific keywords that are instrumental to task success: colors, spatial relations, and other words that are highly indicative of dialogue acts (e.g., responding *"yes"* vs. *"no"*, instructing to *"place"* vs. *"remove"*, etc.). These lists also capture synonyms that are common in our data (e.g. *"yes"*/*"yeah"*), and were obtained by curating non-overlapping lists of words (with a frequency ≥ 10 across all data splits) that are appropriate to each category.[2]

We report precision and recall scores per category, and for an "all keywords" list consisting of the union of all category word lists. For each category, we reduce both human and generated utterances to those tokens that occur in the corresponding keyword list: *"place another red left of the green"* reduces to *"red green"* for color, to *"left"* for spatial relations and *"place"* for dialogue.

For a given (reduced) generated sentence S_g and its associated (reduced) human utterance S_h, we calculate term-specific precision (and recall) as follows. Any token t_g in S_g matches a token t_h in S_h if t_g and t_h are identical or synonyms. Similar to BLEU's modified unigram precision, once t_g is matched to one token t_h, it cannot be used for further matches to other tokens within S_h. Counts are accumulated over the entire corpus to compute the ratio of matched to total tokens in S_g (or S_h).

Ablation study Table 1 shows the results of an ablation study on the validation set. All model variants here share the same RNN parameters. While the individual addition of global and local block counters each see a slight boost in performance in precision and recall respectively, combining them as in our final model shows significant performance increase, especially on colors.

Test set results We finetune our most basic and most complex model via a grid search over all architectural parameters and dropout values on the validation set. The best model's results on the test set are shown in Table 2. Our full model shows noticeable improvements on each of our metrics over the baseline. Most promising is again the significant increase in performance on colors, indicating that the block counters capture necessary information about next Builder actions.

9.2 Human Evaluation

In order to better evaluate the quality of generated utterances as well as benchmark human performance, we performed a small-scale human evaluation of Architect utterances. We asked 3 hu-

[2] These word lists are in the supplementary materials.

Metric	BLEU B-1	B-2	B-3	B-4	Precision / Recall all keywords	colors	spatial	dialogue
seq2seq	14.9	6.9	3.8	2.1	12.0 / 10.3	8.4 / 12.1	9.9 / 9.1	16.5 / 19.1
+ global only	16.1	7.7	4.1	2.4	12.9 / 11.6	14.4 / 15.5	8.8 / 7.0	19.1 / 18.8
+ local only	16.0	7.9	4.5	2.6	13.5 / **13.8**	13.3 / **23.5**	9.5 / **11.3**	**19.3 / 22.0**
+ global & local	**16.2**	**8.1**	**4.7**	**2.8**	**14.5 / 13.8**	**14.8** / 23.3	**10.7** / 9.5	17.9 / 20.6

Table 1: BLEU score and term-specific precision and recall ablation study on the validation set.

Metric	BLEU B-1	B-2	B-3	B-4	Precision / Recall all keywords	colors	spatial	dialogue
seq2seq	15.3	7.8	4.5	2.8	11.8 / 11.1	8.1 / 17.0	**9.3** / 8.6	17.9 / 19.3
+ global & local	**15.7**	**8.1**	**4.8**	**2.9**	**13.5 / 14.4**	**14.9 / 28.7**	8.7 / **8.7**	**18.5 / 19.9**

Table 2: BLEU and term-specific precision and recall scores of the seq2seq and the full model on the test set.

man participants who had previously completed the Minecraft Collaborative Building Task to evaluate 100 randomly sampled scenarios from the test set. Each scenario was reenacted from an actual human-human game by simulating the context of dialogue and Builder actions in Minecraft. Then, we presented 3 candidate Architect utterances to follow that context (one each generated from the models in Table 2 as well as the original human utterance) to the evaluators in randomized order.

Here, we analyze a subset of results on coarse annotation of dialogue acts and utterance correctness. More details on the full evaluation framework, including descriptions of evaluation criteria and inter-annotator agreement statistics, are included in the supplementary materials.

Dialogue acts Given a list of six predefined coarse-grained dialogue acts (including *Instruct **B***, *Describe Target*, etc.; see the supplementary material for full details), evaluators were asked to choose all dialogue acts that categorized a candidate utterance. An utterance could belong to any number of categories; e.g., *"great! now place a red block"* is both a confirmation as well as an instruction. Results can be found in Table 3. These results show a significantly higher diversity of utterance types generated by humans. Humans provided instructions only about half of the time, and devoted more energy to providing higher-level descriptions of the target, responding to the Builder's actions and queries, and rectifying mistakes. On the other hand, even the improved model failed to capture this, mainly generating instructions even if it was inappropriate or unhelpful to do so.

Utterance correctness Given a window of game context (consisting of at least the last seven Builder's and Architect's actions, but always including the previous Architect's utterance) and access to the target structure to be built, evaluators were asked to rate the correctness of an utterance immediately following that context with respect to task completion. For an utterance to be *fully* correct, information contained within it must both be consistent with the current state of the world as well as not lead the Builder off-course from the target. Utterances could be considered *partially* correct if some described elements (e.g. colors) were accurate, but other incorrect elements precluded full correctness. Otherwise, utterances could be deemed *incorrect* (if wildly off-course) or *N/A* (if there was not enough information). Results can be found in Table 4. Unsurprisingly, without access to world state information, the baseline model performs poorly, conveying incorrect information about half of the time. With access to a simple world representation, our full model shows marked improvement on generating both *fully* and *partially* correct utterances. Finally, human performance sets a high bar; when not engaging in chitchat or correcting typos, humans consistently produce fully correct utterances constructive towards task completion.

9.3 Qualitative Analysis

Here, we use examples to illustrate different aspects of our best model's utterances.

Identifying the game state In the course of a game, players progress through different states. In the human-human data, dialogue is peppered with context cues (greetings, questions, apologies, in-

Model	Instruct **B**	Describe Target	Answer question	Confirm **B**'s actions/plans	Correct/ clarify **A/B**	Other
seq2seq	76.0	12.0	7.0	9.0	3.0	4.0
+ global & local	72.0	14.0	8.0	9.0	3.0	4.0
human	47.0	14.0	12.0	17.0	23.0	8.0

Table 3: Percentage of utterances categorized as a given dialogue act. Labels were determined per dialogue act by majority vote across three human evaluators. An utterance can belong to multiple dialogue acts.

Model	Full	Partial	None	N/A
seq2seq	14.0	28.0	48.0	10.0
+ global & local	25.0	36.0	32.0	7.0
human	89.0	2.0	0.0	9.0

Table 4: Percentage of utterances deemed correct by human evaluators.

structions to move or place blocks) that indicate the flow of a game. Our model is able to capture some of these aspects. It often begins games with an instruction like *"we'll start with blue"*, and may end them with *"ok we're done!"* (although it occasionally continues with further instructions, e.g *"great! now we'll do the same thing on the other side"*.) It often says *"perfect!"* immediately followed by a new instruction which indicates the model's ability to acknowledge a Builder's previous actions before continuing. The model often describes the type of the next required action correctly (even if it makes mistakes in the specifics of that action): it generated *"remove the bottom row"* when the ground truth was *"okay so now get rid of the inner most layer of purple in the square"*.

Predicting block colors and spatial relations Generated utterances often identify the correct color of blocks, e.g *"then place a red block on top of that"* in a context when the the next placements include a layer of red blocks (ground truth utterance: *"the second level of the structure consists wholly of red blocks. start by putting a red block on each orange block"*.) Less frequently, the model is also able to predict accurate spatial relations (*"perfect! now place a red block to the left of that"*) for referent blocks.

Utterance diversity and repetition Generated utterances lack diversity: the pattern *"a x b"* (for a rectangle of size $a \times b$) is almost exclusively used to describe squares (an extremely common shape in our data). Utterances are mostly fluent, but sometimes contain repeats: *"okay, on top of the blue block, put a blue block on top of the blue"* or *"yes, now, purple, purple, purple, ..."*

10 Conclusion and Future Work

The Minecraft Collaborative Building Task provides interesting challenges for interactive agents: they must understand and generate spatially-aware dialogue, execute instructions, identify and recover from mistakes. As a first step towards the goal of developing fully interactive agents for this task, we considered the subtask of Architect utterance generation. To give accurate, high-level instructions, Architects need to align the Builder's world state to the target structure and identify complex substructures. We show that models that capture some world state information improve over naive baselines. Richer models (e.g. CNNs over world states, attention mechanisms (Bahdanau et al., 2015), memory networks (Bordes et al., 2017)) and/or explicit semantic representations should be able to generate better utterances. Clearly, much work remains to be done to create actual agents that can play either role interactively against a human. The Minecraft Dialogue Corpus as well as the Malmo platform and our extension of it enable many such future directions. Our platform can also be extended to support fully interactive scenarios that may involve a human player, measure task completion, or support other training regimes (e.g. reinforcement learning).

Acknowledgements

We would like to thank the reviewers for their valuable comments. This work was supported by Contract W911NF-15-1-0461 with the US Defense Advanced Research Projects Agency (DARPA) Communicating with Computers Program and the Army Research Office (ARO). Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Anne H Anderson, Miles Bader, Ellen Gurman Bard, Elizabeth Boyle, Gwyneth Doherty, Simon Garrod, Stephen Isard, Jacqueline Kowtko, Jan McAllister, Jim Miller, et al. 1991. The HCRC map task corpus. *Language and speech*, 34(4):351–366.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Yonatan Bisk, Daniel Marcu, and William Wong. 2016a. Towards a dataset for human computer communication via grounded language acquisition. In *AAAI Workshop: Symbiotic Cognitive Systems*.

Yonatan Bisk, Kevin Shih, Yejin Choi, and Daniel Marcu. 2018. Learning interpretable spatial operations in a rich 3D Blocks World. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5028–5036.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 2016b. Natural language communication with robots. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 751–761, San Diego, California. Association for Computational Linguistics.

Antoine Bordes, Y.-Lan Boureau, and Jason Weston. 2017. Learning end-to-end goal-oriented dialog. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - a large-scale multi-domain wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.

Joyce Y. Chai, Qiaozi Gao, Lanbo She, Shaohua Yang, Sari Saba-Sadiya, and Guangyue Xu. 2018. Language to action: Towards interactive task learning with physical agents. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 2–9. International Joint Conferences on Artificial Intelligence Organization.

David Chen and Raymond Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 859–865.

Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José M.F. Moura, Devi Parikh, and Dhruv Batra. 2017. Visual Dialog. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 326–335.

Srinivasan Janarthanam, Oliver Lemon, and Xingkun Liu. 2012. A web-based evaluation framework for spatial instruction-giving systems. In *Proceedings of the ACL 2012 System Demonstrations*, pages 49–54, Jeju Island, Korea. Association for Computational Linguistics.

Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. 2016. The Malmo platform for artificial intelligence experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, pages 4246–4247.

Seokhwan Kim, Luis Fernando D'Haro, Rafael E Banchs, Jason D Williams, and Matthew Henderson. 2017. The fourth dialog state tracking challenge. In *Dialogues with Social Robots*, pages 435–449. Springer.

Seokhwan Kim, Luis Fernando D'Haro, Rafael E Banchs, Jason D Williams, Matthew Henderson, and Koichiro Yoshino. 2016. The fifth dialog state tracking challenge. In *2016 IEEE Spoken Language Technology Workshop (SLT)*, pages 511–517.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Alexander Koller, Kristina Striegnitz, Donna Byron, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. 2010. The first challenge on generating instructions in virtual environments. In *Empirical Methods in Natural Language Generation*, pages 328–352, Berlin, Heidelberg. Springer-Verlag.

Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. A simple, fast diverse decoding algorithm for neural generation. *arXiv preprint arXiv:1611.08562*.

Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 2015. The Ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 285–294, Prague, Czech Republic. Association for Computational Linguistics.

Dipendra K. Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. 2016. Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Ramakanth Pasunuru and Mohit Bansal. 2018. Game-based video-context dialogue. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 125–136, Brussels, Belgium. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Alan Ritter, Colin Cherry, and Bill Dolan. 2010. Unsupervised modeling of Twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 172–180, Los Angeles, California. Association for Computational Linguistics.

Nicolas Schrading, Cecilia Ovesdotter Alm, Ray Ptucha, and Christopher Homan. 2015. An analysis of domestic abuse discourse on Reddit. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2577–2583, Lisbon, Portugal. Association for Computational Linguistics.

M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 1507–1514.

Jesse Thomason, Shiqi Zhang, Raymond J Mooney, and Peter Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 1923–1929.

Sida I. Wang, Samuel Ginn, Percy Liang, and Christopher D. Manning. 2017. Naturalizing a programming language via interactive learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 929–938, Vancouver, Canada. Association for Computational Linguistics.

Terry Winograd. 1971. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MIT. Cent. Space Res.