

CraftAssist Instruction Parsing: Semantic Parsing for a Minecraft Assistant

Anonymous ACL submission

Abstract

We propose a semantic parsing dataset focused on instruction-driven communication with an agent in the game Minecraft. The dataset consists of 7K human utterances and their corresponding parses. Given proper world state, the parses can be interpreted and executed in game. We report the performance of baseline models, and analyze their successes and failures.

1 Introduction

Semantic parsing is used as a component for natural language understanding in human-robot interaction systems (Lauria et al., 2001; Bos and Oka, 2007; Tellex et al., 2011; Matuszek et al., 2013; Thomason et al., 2019), and for virtual assistants (Kollar et al., 2018). We would like to be able to apply deep learning methods in this space, as recently researchers have shown success with these methods for semantic parsing more generally, e.g. (Dong and Lapata, 2016; Jia and Liang, 2016; Zhong et al., 2017). However, to fully utilize powerful neural network approaches, it is necessary to have large numbers of training examples. In the space of human-robot (or human-assistant) interaction, the publicly available semantic parsing datasets are small. Furthermore, it can be difficult to reproduce the end-to-end results (from utterance to action in the environment) because of the wide variety of robot setups and proprietary nature of personal assistants.

In this work, we introduce a new semantic parsing dataset for human-bot interactions. Our “robot” or “assistant” is embodied in the sandbox construction game Minecraft¹, a popular multiplayer open-world voxel-based crafting game. We also provide the associated platform for executing the logical forms in game.

¹<https://minecraft.net/en-us/>. We limit ourselves to creative mode for this work

Situating the assistant in Minecraft has several benefits for studying task oriented natural language understanding (NLU). Compared to physical robots, Minecraft allows less technical overhead irrelevant to NLU, such as difficulties with hardware and large scale data collection. On the other hand, our bot has all the basic in-game capabilities of a player, including movement and placing or removing voxels. Thus Minecraft preserves many of the NLU elements of physical robots, such as discussions of navigation and spatial object reference.

Working in Minecraft may enable large scale human interaction because of its large player base, in the tens of millions. Furthermore, although Minecraft’s simulation of physics is simplified, the task space is complex. While there are many atomic objects in the game, such as animals and block-types, that require no perceptual modeling, the player also interacts with complex structures made up of collections of voxels such as a “house” or a “hill”. The assistant cannot apprehend them without a perceptual system, creating an ideal test bed for researchers interested in the interactions between perception and language.

Our contributions in the paper are as follows:
Grammar: We develop a grammar over a set of primitives that comprise a mid-level interface to Minecraft for machine learning agents.

Data: We collect 7K crowd-sourced annotations of commands generated independent of our grammar. In addition to the natural language commands and the associated logical forms, we release the tools used to collect these, which allow crowd-workers to efficiently and accurately annotate parses.

Models: We show the results of several neural semantic parsing models trained on our data.

Execution: Finally, we also make available the code to execute logical forms in the game, allow-

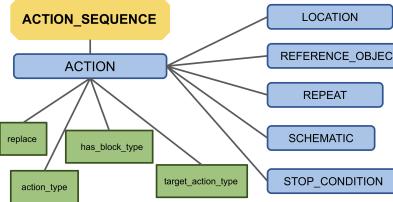


Figure 1: The basic structure of the ACTION_SEQUENCE branch of the assistant’s grammar. The gold octagon is an internal node whose children are ordered, blue rectangles are regular internal nodes, and green rectangles are categorical leaf nodes. Not all combinations of children of ACTION are possible, see the full list of possible productions (and the productions for PUT_MEMORY and GET_MEMORY) in the Appendix.

ing the reproduction of end-to-end results. This also opens the door to using the data for reinforcement and imitation learning with language. We also provide access to an interactive bot using these models for parsing².

2 The Assistant Grammar

In this section we summarize a grammar for generating logical forms that can be interpreted into programs for the agent architecture described in (Gray et al., 2019).

2.1 Agent Action Space

The assistant’s basic functions include moving, and placing and destroying blocks. Supporting these basic functions are methods for control flow and memory manipulation.

Basic action commands: The assistant can MOVE to a specified location; or DANCE with a specified sequence of steps. It can BUILD an object from a known schematic (or by making a copy of a block-object in the world) at a given location, or DESTROY an existing object. It can DIG a hole of a given shape at a specified location, or FILL one up. The agent can also be asked to complete a partially built structure however it sees fit by FREEBUILD. Finally, it can SPAWN a mob (an animate NPC in Minecraft).

Control commands: Additionally, the agent can STOP or RESUME an action, or UNDO the result of a recent command. Furthermore, the assistant can LOOP given a task and a stop-condition. Finally, it needs to be able to understand when a sentence

²Instructions can be found at <http://craftassist.io/acl2020demo>, requires a Minecraft license and client.

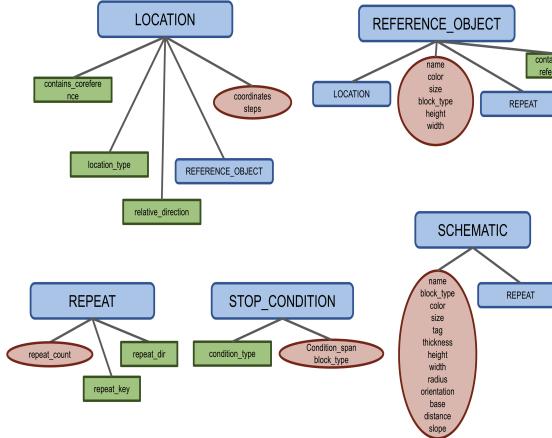


Figure 2: The basic structure of internal nodes in the assistant’s grammar. Blue rectangles are internal nodes, green rectangles are categorical leaf nodes, and red ovals are span nodes.

does not correspond to any of the above mentioned actions, and map it to a NOOP.

Memory interface: Finally, the assistant can interact with its SQL based memory. It can place or update rows or cells, for example for tagging objects. This can be considered a basic version of the self-improvement capabilities in (Kollar et al., 2013; Thomason et al., 2015; Wang et al., 2016, 2017). It can retrieve information for question answering similar to the VQA in (Yi et al., 2018).

2.2 Logical Forms

The focus of this paper is an intermediate representation that allows natural language to be interpreted into programs over the basic actions from the previous section. The logical forms (represented as trees) making up this representation consist of three basic types of nodes: “internal nodes” that can have children, “categorical” (leaf) nodes that belong to a fixed set of possibilities, and “span” nodes that point to a region of text in the natural language utterance. The full grammar is shown in the Appendix; and a partial schematic representation is shown in Figures 1 and 2. In the paragraphs below, we give more detail about some of the kinds of nodes in the grammar.

We emphasize that this is an *intermediate* representation. The logical forms do not come with any mechanism for generating language, and nodes do not correspond in any simple way with words. On the other hand, the logical forms do not encode all of the information necessary for execution without the use of an interpreter that can access the

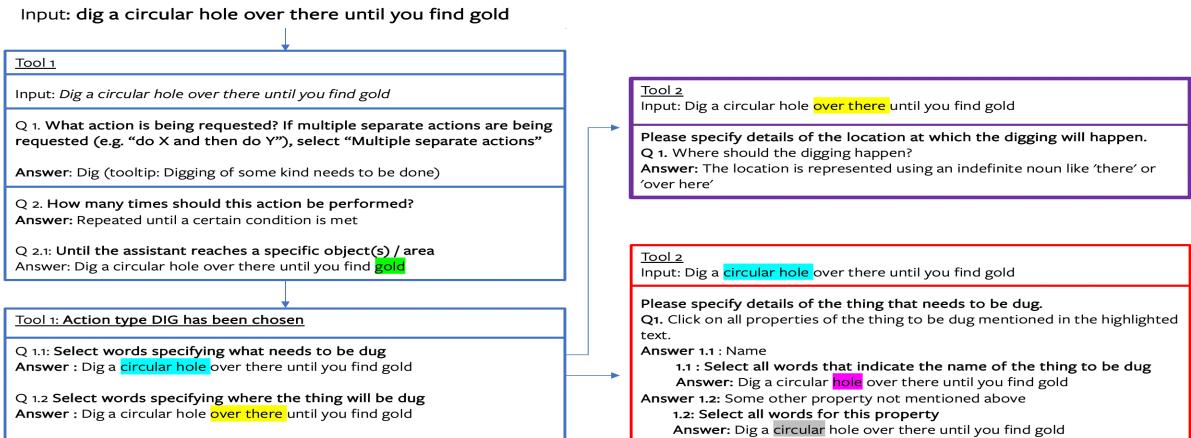


Figure 3: A representation of the annotation process using the web-based annotation tool described in Section 3.1.3. The colors of the boxes correspond to annotation tasks. The highlighting on the text in the header of the later tasks is provided by a previous annotator. We show more detailed screenshots of how the tool works in Appendix B.3

assistant’s memory and the Minecraft world state.

Internal nodes: Internal nodes are nodes that allow recursion; although most do not require it. They can correspond to top-level actions, for example BUILD; in which case they would just be an “action” node with “action_type” build; see Figure 1. They can also correspond to arguments to top-level actions, for example a “reference_object”, which specifies an object that has a spatial location. Internal nodes are not generally required to have children; it is the job of the interpreter to deal with under-specified programs like a BUILD with no arguments.

In addition to the various LOCATION, REFERENCE OBJECT, SCHEMATIC, and REPEAT nodes which can be found at various levels, another notable sub-tree is the action’s STOP CONDITION, which essentially allows the agent to understand “while” loops (for example: “dig down until you hit the bedrock” or “follow me”).

Leaf nodes: Eventually, arguments have to be specified in terms of values which correspond to (fixed) agent primitives. We call these nodes categorical leaves (green rectangles in Figures 1 and 2). As mentioned above, an “action” internal node has a categorical leaf child which specifies the **action type**. There are also **repeat type** nodes similarly specifying a kind of loop for example in the REPEAT sub-tree corresponding to “make three houses” the **repeat type for** specifies a “for” loop). There are also **location type** nodes specifying if a location is determined by a reference object, a set

of coordinates, etc.; **relative direction** nodes that have values like “left” or “right”. The complete list of categorical nodes is given in the Appendix.

However, there are limits to what we can represent with a pre-specified set of hard-coded primitives, especially if we want our agent to be able to learn new concepts or new values. Additionally, even when there is a pre-specified agent primitive, mapping some parts of the command to a specific value might be better left to an external module (e.g. mapping a number string to an integer value). For these reasons, we also have span leaves (red ovals in Figure 2). For example, in the parse for the command “*Make three oak wood houses to the left of the dark grey church.*”, the SCHEMATIC (an internal node) might be specified by the command sub-string corresponding to its **name** by the span “houses” and the requested **block type** by the span “oak wood”. The range of the for loop is specified by the REPEAT’s **for value** (“three”), and the REFERENCE OBJECT for the location is denoted in the command by its generic **name** and specific **color** with spans “church” and “dark grey”.

The root: The root of the tree has three productions: PUT_MEMORY, and GET_MEMORY, corresponding to writing to memory and reading from memory; and HUMAN_GIVE_COMMAND which also produces an ACTION_SEQUENCE, which is a special internal node whose children are ordered; multiple children correspond to an ordered sequence of commands (“build a house and then a tower”). In Figures 1 and 2 we show a schematic

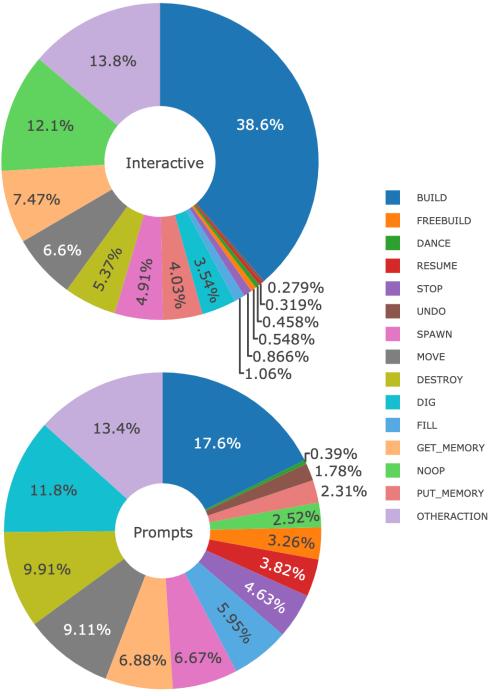


Figure 4: Frequency of each action type in the different data collection schemes described in Section 3.1.

representation for an ACTION_SEQUENCE.

3 The CAIP Dataset

This paper introduces the CraftAssist Instruction Parsing (CAIP) dataset of English-language commands and their associated logical forms (see Appendix C for examples and a full grammar specification).

3.1 Collected Data

We collected natural language commands written by crowd-sourced workers in a variety of settings. The complete list of instructions given to crowd-workers in different settings, as well as step-by-step screen-shot of the annotation tool, are provided in the Appendix B. The basic data cleanup is described in Appendix A.

3.1.1 Image and Text Prompts

We presented crowd-sourced workers with a description of the capabilities of an assistant bot in a creative virtual environment (which matches the set of allowed actions in the grammar), and (optionally) some images of a bot in a game environment. They were then asked to provide examples of commands that they might issue to an in-game assistant. We refer to these instructions as “prompts” in the rest of this paper.

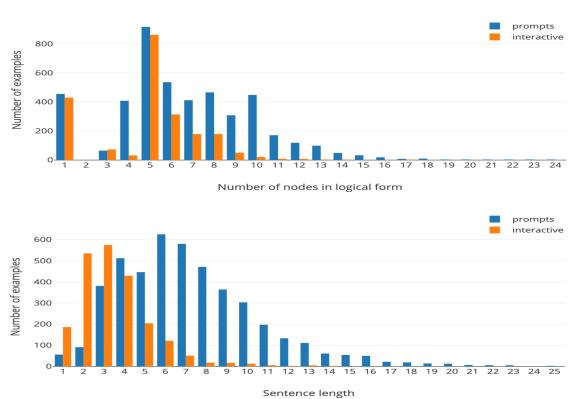


Figure 5: Histograms showing distribution over number of nodes in a logical form (top) and utterance length in words (bottom) for each data type. Prompts averages 6.74 nodes per logical form, 7.32 words per utterance, and interactive averages 4.89, 3.42 respectively

3.1.2 Interactive Gameplay

We asked crowd-workers to play creative-mode Minecraft with our assistant bot, and they were instructed to use the in-game chat to direct the bot as they chose. The game sessions were capped at 10 minutes and players in this setting had no prior knowledge of the bot’s capabilities or the grammar. We refer to these instructions as “Interactive” in the rest of this paper. The instructions of this setting are included in Appendix B.2.

3.1.3 Annotation Tool

Both prompts and interactive instructions come without a reference logical form and need to be annotated. To facilitate this process, we designed a multi-step web-based tool which asks users a series of multiple-choice questions to determine the semantic content of a sentence. The responses to some questions will prompt other more specific questions, in a process that mirrors the hierarchical structure of the grammar. The responses are then processed to produce the complete logical form. This allows crowd-workers to provide annotations with no knowledge of the specifics of the grammar described above. A pictorial representation of the annotation process is shown in Figure 3 and a more detailed explanation of the process along with screen-shots of the tool is given in Appendix B.3.

We used a small set of tasks that were representative of the actual annotations to select skilled crowd-sourced workers by manually verifying the accuracy of responses on these.

Each utterance in our collection of prompts and

interactive chats was shown to three different qualified annotators and we included the utterance and logical form in the dataset only if at least 2 out of 3 qualified annotators agreed on the logical form output. The total number of utterances sent to turkers was 6,775. Out of these, 6,693 had at least 2/3 agreements on the logical form and were kept. Of these, 2,872 had 3/3 agreements.

The final dataset has 4,532 annotated instructions from the prompts setting (Section 3.1.1), and 2,161 from interactive play (Section 3.1.2). The exact instructions shown to Turkers in the annotation tools are reproduced in Figures 9 and 11 in supplementary.

As in (Yih et al., 2016), we have found that careful design of the annotation tool leads to significant improvements in efficiency and accuracy. In particular, we re-affirm the conclusion from (Yih et al., 2016) that having each worker do one task (e.g. labeling a single node in the tree) makes annotation easier for workers.

3.2 Dataset Statistics

3.2.1 Action Frequencies

Since the different data collection settings described in Section 3.1 imposed different constraints and biases on the crowd-sourced workers, the distribution of actions in each subset of data is therefore different. The action frequencies of each subset are shown in Figure 4.

3.2.2 Grammar coverage

Some crowd-sourced commands describe an action that is outside the scope of the grammar. To account for this, users of the annotation tool are able to mark that a sentence is a command to perform an action that is not covered by our grammar yet. The resulting trees are labeled as OTHERACTION, and their frequency in each dataset is shown in Figure 4. Annotators still have the option to label other nodes in the tree, such as the action’s LOCATION or REFERENCE OBJECT. In both the prompts and interactive data, OTHERACTION amounted to approximately 14% of the data.

3.2.3 Quantitative analysis

For each of our data types, Figure 5 show a histogram of sentence length and number of nodes. On an average interactive data has shorter sentences and smaller trees.

3.2.4 Qualitative Linguistic Style

We show the linguistic styles and choice of words of the data sources by displaying the surface forms of a set of trees. We randomly picked trees of size (number of nodes) 7 that appear in both data sources, and then for the same tree structure, we looked at the utterances corresponding to that tree. We show some representative examples in table 1. We show more examples of the data in the Appendix.

4 Related Work

There have been a number of datasets of natural language paired with logical forms to evaluate semantic parsing approaches, e.g. (Price, 1990; Tang and Mooney, 2001; Cai and Yates, 2013; Wang et al., 2015; Zhong et al., 2017). The dataset presented in this work is an order of magnitude larger than those in (Price, 1990; Tang and Mooney, 2001; Cai and Yates, 2013) and is similar in scale to the datasets in (Wang et al., 2015), but smaller than (Zhong et al., 2017).

In addition to mapping natural language to logical forms, our dataset connects both of these to a dynamic environment. In (Lauria et al., 2001; Bos and Oka, 2007; Tellez et al., 2011; Matuszek et al., 2013; Thomason et al., 2019) semantic parsing has been used for interpreting natural language commands for robots. In our paper, the “robot” is embodied in the Minecraft game instead of in the physical world. In (Boye et al., 2006) semantic parsing has been used for spoken dialogue with an embodied character in a 3-D world with pattern matching and rewriting phases. In our work, the user along with the assistant is embodied in game and instructs using language. We go from language to logical forms end-to-end with no pattern match necessary. Semantic parsing in a voxel-world recalls (Wang et al., 2017), where the authors describe a method for building up a programming language from a small core via interactions with players.

We demonstrate the results of several neural parsing models on our dataset. In particular, we show the results of a re-implementation of (Dong and Lapata, 2016) adapted to our grammar, and a straightforward fine-tuned BERT model (Devlin et al., 2018). There have been several other papers proposing neural architectures for semantic parsing, for example (Jia and Liang, 2016; Zhong et al., 2017; Wang et al., 2018; Hwang et al., 2019);

500	Prompts	bot move to where the tree is	dig a large size hole to put these waste particles into the hole	please build a sphere on that location	hey bot can you dig a 5 by 5 hole for me	550
502	Interactive	find tree	dig large hole	build a sphere over here	dig a 5 x 5 hole	552

503
504
505
Table 1: Choice of words across different data sources for the same logical form (per column).
553
554
555

506 in particular (Hwang et al., 2019) uses a BERT
507 based model. In those papers, as in this one, the
508 models are trained with full supervision of the map-
509 ping from natural language to logical forms, with-
510 out considering the results of executing the logical
511 form (in this case, the effect on the environment of
512 executing the actions denoted by the logical form).
513 There has been progress towards “weakly super-
514 vised” semantic parsing (Artzi and Zettlemoyer,
515 2013; Liang et al., 2016; Guu et al., 2017) where
516 the logical forms are hidden variables, and the only
517 supervision given is the result of executing the log-
518 ical form. There are now approaches that have
519 shown promise without even passing through (dis-
520 crete) logical forms at all (Riedel et al., 2016; Nee-
521 lakantan et al., 2016). We hope that the dataset
522 introduced here, which has supervision at the level
523 of the logical forms, but whose underlying gram-
524 mar and environment can be used to generate es-
525 sentially infinite weakly supervised or execution
526 rewards, will also be useful for studying these mod-
527 els.

528 Minecraft, especially via the MALMO project
529 (Johnson et al., 2016) has been used as a base en-
530 vironment for several machine learning papers. It
531 is often used as a testbed for reinforcement learn-
532 ing (RL) (Shu et al., 2017; Udagawa et al., 2016;
533 Alaniz, 2018; Oh et al., 2016; Tessler et al., 2017).
534 In these works, the agent is trained to complete
535 tasks by issuing low level actions (as opposed to our
536 higher level primitives) and receiving a reward on
537 success. Others have collected large-scale datasets
538 for RL and imitation learning (Guss et al., 2019a,b).
539 Some of these works (e.g. (Oh et al., 2017)) do con-
540 sider simplified, templated language as a method
541 for composable specifying tasks, but training an
542 RL agent to execute the scripted primitives in our
543 grammar is already nontrivial, and so the task space
544 and language in those works is more constrained
545 than what we use here. Nevertheless, our work
546 may be useful to researchers interested in RL (or
547 imitation): using our grammar and executing in
548 game can supply (hard) tasks and descriptions, and
549 demonstrations. Another set of works (Kitaev and
Klein, 2017; Yi et al., 2018) have used Minecraft

556 for visual question answering with logical forms.
557 Our work extends these to interactions with the en-
558 vironment. Finally, (Allison et al., 2018) is a more
559 focused study on how a human might interact with
560 a Minecraft agent; our collection of free genera-
561 tions (see 3.1.1) includes annotated examples from
562 similar studies of players interacting with a player
563 pretending to be a bot.

5 Baseline Models

In order to assess the challenges of the dataset,
we implement two models which learn to read a
sentence and output a logical form by formulating
the problem as a sequence-to-tree and a sequence-
to-sequence prediction task respectively.

5.1 Sequence to Tree Model

Our first model adapts the Seq2Tree approach of
(Dong and Lapata, 2016) to our grammar. In short,
a bidirectional RNN encodes the input sentence
into a sequence of vectors, and a decoder recur-
sively predicts the tree representation of the logical
form, starting at the root and predicting all of the
children of each node based on its parent and left
siblings and input representation.

Sentence Encoder and Attention: We use a
bidirectional GRU encoder (Cho et al., 2014) which
encodes a sentence of length T $\mathbf{s} = (w_1, \dots, w_T)$
into a sequence of T dimension d vectors:

$$f_{GRU}(\mathbf{s}) = (\mathbf{h}_1, \dots, \mathbf{h}_T) \in \mathbb{R}^{d \times T}$$

Tree Decoder: The decoder starts at the root,
computes its node representation and predicts the
state of its children, then recursively computes the
representations of the predicted descendants. Simi-
larly to Seq2Tree, a node representation \mathbf{r}_{nt} is com-
puted based on its ancestors and left siblings. We
also found it useful to condition each of the node
representation on the encoder output explicitly for
each node. Thus, we compute the representation
 \mathbf{r}_{nt} and recurrent hidden state \mathbf{g}_{nt} for node nt as:

$$\mathbf{r}_{nt} = \text{attn}(\mathbf{v}_{nt} + \mathbf{g}_{nt-1}, (\mathbf{h}_1, \dots, \mathbf{h}_T); \mathbf{M}^\sigma) \quad (1)$$

$$\mathbf{g}_{nt} = f_{rec}(\mathbf{g}_{nt-1}, (\mathbf{v}'_{nt} + \mathbf{r}_{nt})) \quad (2)$$

Where attn is multi-head attention, $\mathbf{M}^\sigma \in \mathbb{R}^{d \times d \times K}$ is a tree-wise parameter, f_{rec} is the GRU recurrence function, and \mathbf{v}'_{n_t} is a node parameter (one per category for categorical nodes), and n_{t-1} denotes either the last predicted left sibling if there is one or the parent node otherwise.

Prediction Heads: Finally, the decoder uses the computed node representations to predict the state of each of the internal, categorical, and span nodes in the grammar. We denote each of these sets by \mathcal{I} , \mathcal{C} and \mathcal{S} respectively, and the full set of nodes as $\mathcal{N} = \mathcal{I} \cup \mathcal{C} \cup \mathcal{S}$.

First, each node in \mathcal{N} is either active or inactive in a specific logical form. We denote the state of a node n by $a_n \in \{0, 1\}$. All the descendants of an inactive internal node $n \in \mathcal{I}$ are considered to be inactive. Additionally, each categorical node $n \in \mathcal{C}$ has a set of possible values C^n ; its value in a specific logical form is denoted by the category label $c_n \in \{1, \dots, |C^n|\}$. Finally, active span nodes $n \in \mathcal{S}$ for a sentence of length T have a start and end index $(s_n, e_n) \in \{1, \dots, T\}^2$. We compute, the representations \mathbf{r}_n of the nodes as outlined above, then obtain the probabilities of each of the labels by:

$$\forall n \in \mathcal{N}, p(a_n) = \sigma(\langle \mathbf{r}_n, \mathbf{p}_n \rangle) \quad (3)$$

$$\forall n \in \mathcal{C}, p(c_n) = \text{softmax}(M_n^c \mathbf{r}_n) \quad (4)$$

$$\begin{aligned} \forall n \in \mathcal{S}, p(s_n) &= \text{softmax}(\mathbf{r}_n^T M_n^s (\mathbf{h}_1, \dots, \mathbf{h}_T)) \\ p(e_n) &= \text{softmax}(\mathbf{r}_n^T M_n^e (\mathbf{h}_1, \dots, \mathbf{h}_T)) \end{aligned} \quad (5)$$

where the following are model parameters:

$$\forall n \in \mathcal{N}, \mathbf{p}_n \in \mathbb{R}^d$$

$$\forall n \in \mathcal{C}, M_n^c \in \mathbb{R}^{d \times d}$$

$$\forall n \in \mathcal{S}, (M_n^s, M_n^e)_n \in \mathbb{R}^{d \times d \times 2}$$

Let us note the parent of a node n as $\pi(n)$. Given Equations 3 to 5, the log-likelihood of a tree with states $(\mathbf{a}, \mathbf{c}, \mathbf{s}, \mathbf{e})$ given a sentence \mathbf{s} is then:

$$\begin{aligned} \mathcal{L} = & \sum_{n \in \mathcal{N}} a_{\pi(n)} \log(p(a_n)) + \sum_{n \in \mathcal{C}} a_n \log(p(c_n)) \\ & + \sum_{n \in \mathcal{S}} a_n (\log(p(s_n)) + \log(p(e_n))) \end{aligned} \quad (6)$$

Overall, our implementation differs from the original Seq2Tree in three ways, which we found lead to better performance in our setting. First, we replace single-head with multi-head attention.

Secondly, the cross-attention between the decoder and attention is conditioned on both the node embedding and previous recurrent state. Finally, we replace the categorical prediction of the next node by a binary prediction problem: since we know which nodes are eligible as the children of a specific node (see Figures 1 and 2), we find that this enforces a stronger prior. We refer to this modified implementation as SentenceRec.

5.2 Sequence to Sequence Model

Our second approach treats the problem of predicting the logical form as a general sequence-to-sequence (Seq2Seq) task; such approaches have been used in semantic parsing in e.g. (Jia and Liang, 2016; Wang et al., 2018). We take the approach of (Jia and Liang, 2016) and linearize the output trees: the target sequence corresponds to a Depth First Search walk through the tree representation of the logical form. More specifically the model needs to predict, in DFS order, a sequence of tokens corresponding to opening and closing internal nodes, categorical leaves and their value, and span leaves with start and end sequences. In practice, we let the model predict span nodes in two steps: first predict the presence of the node, then predict the span value, using the same prediction heads as for the SentenceRec model (see Equation 5 above). With this formalism, the logical form for e.g. “build a large blue dome on top of the walls” will be:

```
(ACTION_TYPE:BUILD, OPEN:SCHEMATIC,
HAS_SIZE, SIZE_SPAN-(2, 2),
HAS_COLOR, COLOR_SPAN-(3, 3),
HAS_NAME, NAME_SPAN-(4, 4),
CLOSE:SCHEMATIC, OPEN:LOCATION,
LOC_TYPE:REF_OBJECT, REL_DIR:UP,
OPEN:REF_OBJECT,
HAS_NAME, NAME_SPAN-(9, 9),
CLOSE:REF_OBJECT,
CLOSE:LOCATION)
```

We train a BERT encoder-decoder architecture on this sequence transduction task, where the training loss is a convex combination of the output sequence log-likelihood and the span cross-entropy loss.

Pre-trained Sentence Encoder: Finally, recent work has shown that using sentence encoder that has been pre-trained on large-scale language modeling tasks can lead to substantial performance improvements (Song et al., 2019). We use the pre-trained DistilBERT model of (Sanh et al., 2019) as the encoder of our sequence-to-sequence model, and also propose a version of the SentenceRec

	Acc. (std)	Inter.	Prompts
SentRec	50.08 (2.97)	64.17	42.49
DistBERT+SentRec	59.58 (3.49)	76.0	50.74
DistBERT+Seq2Seq	60.74 (3.58)	76.06	52.49

Table 2: Average accuracy over a test set of 650 Prompts + 350 Interactive.

	N=2	N=5	N=15
Joint	67.7	72.76	75.7
Interactive	83.83	88.34	90.63
Prompts	59.02	64.37	67.66

Table 3: Recall at N for the Seq2Seq model beam search.

which uses it to replace the bidirectional RNN.

6 Experiments

In this Section, we evaluate the performance of our baseline models on the proposed dataset.

Training Data: The CAIP datasets consists in a total of 6693 annotated instruction-parse pairs. In order for our models to make the most of this data while keeping the evaluation statistically significant, we create 5 different train/test splits of the data and report the average performance of models trained and evaluated on each of them. In each case, we hold out 650 examples from Prompts and 350 from Interactive for testing, and use the remaining 5693 as the training set.

Modeling Choices: For the end-to-end trained SentenceRec model, we use a 2-layer GRU sentence encoder and all hidden layers have dimension $d = 256$. We use pre-trained word embeddings computed with FastText with subword information (Bojanowski et al., 2017). The decoder uses a GRU recurrent cell and 4-headed attention. The Seq2Seq model uses a variant of the *bert-base-uncased* provided in the Transformer library³ with 6 encoding and decoding layers. For the Seq2Seq model and the SentenceRec with pre-trained encoder, we use the *distilbert-base-uncased* encoder from the same library. The Seq2Seq model uses beam search decoding with 15 beams. All models are trained with the Adam optimizer with quadratic learning rate decay. We provide our model and training code along with the dataset for reproducibility purposes.

Overview of Results: Table 2 provides the average accuracy (computed as the proportion of logical

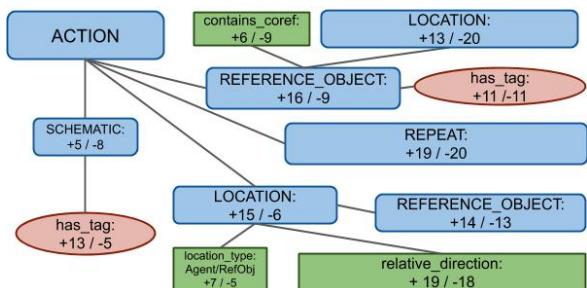


Figure 6: We show nodes in the grammar which are most often wrongly predicted, with false positive (+) and false negative counts (-).

forms that are entirely accurately predicted) and standard deviation across all five splits, as well as the contributions of the Interactive and Prompts data. The first observation is that using a pre-trained encoder leads to a significant improvement, with a 10 point boost in accuracy. On the other hand, while the Seq2Seq model is more general and makes less use of our prior knowledge of the structure of logical forms, it does marginally better than the recursive prediction model (although within one standard deviation).

Secondly, although the models are trained on more data provided from the Prompts setting than from Interactive play, they all do better on the latter. This is consistent with previous observations on the dataset statistics in Section 3.2.3 which find that players tend to give shorter instructions with simpler execution. Finally, we note that one of the advantages of having the parser be part of an interactive agent is that it can ask the player for clarification and adapt its behavior when it is made aware of a mistake (Yao et al., 2019). In that spirit, Table 3 provides Recall at N numbers, which represent how often the true parse is within the N first elements of the beam after beam search. Recall at 2 does provide a consistent boost over the accuracy of a single prediction, but even the full size 15 beam does not always contain the right logical form.

Error Analysis: We further investigate the errors of the Seq2seq models on one of the data splits. We find that the model still struggles with span predictions: out of 363 errors, 125 only make mistakes on spans (and 199 get the tree structure right but make mistakes on leaves). Figure 6 shows the nodes which are most commonly mistaken, with the number of false positive and false negatives out of these 363 mistakes. Unsurprisingly, the most

³<https://github.com/huggingface/transformers>

800 commonly confused span leaf is “has_tag”, which
 801 we use as a miscellaneous marker. Aside from that
 802 “has_tag” however, the span mistakes are evenly
 803 spread over all other leaves. The next most com-
 804 mon source of mistakes comes from the model
 805 struggling between identifying whether a provided
 806 location corresponds to the target of the action or
 807 to the reference object, and to identify instructions
 808 which imply a repetition. The former indicates
 809 a lack of compositionality in the input representa-
 810 tion: the model correctly identifies that a location is
 811 mentioned, but fails to identify its context. Repeat
 812 conditions on the other hand challenge the model
 813 due to the wide variety of possible stop condition,
 814 a problem we suggest future work pay special at-
 815 tention to.
 816

7 Conclusion

817 In this work, we have described a grammar over a
 818 mid-level interface for a Minecraft assistant. We
 819 then discussed the creation of a dataset of natural
 820 language utterances with associated logical forms
 821 over this grammar that can be executed in-game.
 822 Finally, we showed the results of using this new
 823 dataset to train several neural models for parsing
 824 natural language instructions. Consistently with
 825 recent works, we find that BERT pre-trained mod-
 826 els do better than models trained from scratch, but
 827 there is much space for improvement. We believe
 828 this data will be useful to researchers studying
 829 semantic parsing, especially interactive semantic
 830 parsing, human-robot interaction, and even imita-
 831 tion and reinforcement learning.
 832

833	883
834	884
835	885
836	886
837	887
838	888
839	889
840	890
841	891
842	892
843	893
844	894
845	895
846	896
847	897
848	898
849	899
850	850
851	851
852	852
853	853
854	854
855	855
856	856
857	857
858	858
859	859
860	860
861	861
862	862
863	863
864	864
865	865
866	866
867	867
868	868
869	869
870	870
871	871
872	872
873	873
874	874
875	875
876	876
877	877
878	878
879	879
880	880
881	881
882	882
883	883
884	884
885	885
886	886
887	887
888	888
889	889
890	890
891	891
892	892
893	893
894	894
895	895
896	896
897	897
898	898
899	899

900 References

- 901 Stephan Alaniz. 2018. Deep reinforcement learning
902 with model learning and monte carlo tree search in
903 minecraft. *arXiv preprint arXiv:1803.08456*.
904
- 905 Fraser Allison, Ewa Luger, and Katja Hofmann. 2018.
906 How players speak to an intelligent game character
907 using natural language messages. *Transactions of the
908 Digital Games Research Association*, 4(2).
909
- 910 Yoav Artzi and Luke Zettlemoyer. 2013. Weakly su-
911 pervised learning of semantic parsers for mapping
912 instructions to actions. *Transactions of the Associa-
913 tion for Computational Linguistics*, 1:49–62.
914
- 915 Piotr Bojanowski, Edouard Grave, Armand Joulin, and
916 Tomas Mikolov. 2017. Enriching word vectors with
917 subword information. *TACL*, 5:135–146.
918
- 919 Johan Bos and Tetsushi Oka. 2007. A spoken language
920 interface with a mobile robot. *Artificial Life and
921 Robotics*, 11(1):42–47.
922
- 923 Johan Boye, Joakim Gustafson, and Mats Wirén. 2006.
924 Robust spoken language understanding in a com-
925 puter game. *Speech Commun.*, 48:335–353.
926
- 927 Qingqing Cai and Alexander Yates. 2013. Large-scale
928 semantic parsing via schema matching and lexicon
929 extension. In *Proceedings of the 51st Annual Meet-
930 ing of the Association for Computational Linguistics
931 (Volume 1: Long Papers)*, volume 1, pages 423–433.
932
- 933 Kyunghyun Cho, Bart van Merriënboer, Çağlar
934 Gülcühre, Dzmitry Bahdanau, Fethi Bougares, Hol-
935 ger Schwenk, and Yoshua Bengio. 2014. Learning
936 phrase representations using RNN encoder-decoder
937 for statistical machine translation. In *Proceedings of
938 the 2014 Conference on Empirical Methods in Nat-
939 ural Language Processing, EMNLP 2014, October
940 25-29, 2014, Doha, Qatar, A meeting of SIGDAT,
941 a Special Interest Group of the ACL*, pages 1724–
942 1734.
943
- 944 Jacob Devlin, Ming-Wei Chang, Kenton Lee, and
945 Kristina Toutanova. 2018. Bert: Pre-training of deep
946 bidirectional transformers for language understand-
947 ing. *arXiv preprint arXiv:1810.04805*.
948
- 949 Li Dong and Mirella Lapata. 2016. Language to log-
950 ical form with neural attention. *arXiv preprint
951 arXiv:1601.01280*.
952
- 953 Jonathan Gray, Kavya Srinet, Yacine Jernite, Hao-
954 nan Yu, Zhuoyuan Chen, Demi Guo, Siddharth
955 Goyal, C Lawrence Zitnick, and Arthur Szlam. 2019.
956 Craftassist: A framework for dialogue-enabled inter-
957 active agents. *arXiv preprint arXiv:1907.08584*.
958
- 959 William H Guss, Cayden Codel, Katja Hofmann, Bran-
960 don Houghton, Noboru Kuno, Stephanie Milani,
961 Sharada Mohanty, Diego Perez Liebana, Ruslan
962 Salakhutdinov, Nicholay Topin, et al. 2019a. The
963 minerl competition on sample efficient reinforce-
964 ment learning using human priors. *arXiv preprint
965 arXiv:1904.10079*.
966
- 967 William H Guss, Brandon Houghton, Nicholay Topin,
968 Phillip Wang, Cayden Codel, Manuela Veloso, and
969 Ruslan Salakhutdinov. 2019b. Minerl: a large-scale
970 dataset of minecraft demonstrations. *arXiv preprint
971 arXiv:1907.13440*.
972
- 973 Kelvin Guu, Panupong Pasupat, Evan Zheran Liu,
974 and Percy Liang. 2017. From language to
975 programs: Bridging reinforcement learning and
976 maximum marginal likelihood. *arXiv preprint
977 arXiv:1704.07926*.
978
- 979 Wonseok Hwang, Jinhyung Yim, Seunghyun Park, and
980 Minjoon Seo. 2019. A comprehensive exploration
981 on wikisql with table-aware word contextualization.
982 *arXiv preprint arXiv:1902.01069*.
983
- 984 Robin Jia and Percy Liang. 2016. Data recombi-
985 nation for neural semantic parsing. *arXiv preprint
986 arXiv:1606.03622*.
987
- 988 Matthew Johnson, Katja Hofmann, Tim Hutton, and
989 David Bignell. 2016. The malmo platform for arti-
990 ficial intelligence experimentation. In *IJCAI*, pages
991 4246–4247.
992
- 993 Nikita Kitaev and Dan Klein. 2017. Where is misty?
994 interpreting spatial descriptors by modeling regions
995 in space. In *Proceedings of the 2017 Conference on
996 Empirical Methods in Natural Language Processing*,
997 pages 157–166.
998
- 999 Thomas Kollar, Danielle Berry, Lauren Stuart,
999 Karolina Owczarzak, Tagyoung Chung, Lambert
999 Mathias, Michael Kayser, Bradford Snow, and Spyros
999 Matsoukas. 2018. The alexa meaning represen-
999 tation language. In *Proceedings of the 2018 Con-
999 ference of the North American Chapter of the As-
999 sociation for Computational Linguistics: Human
999 Language Technologies, Volume 3 (Industry Papers)*,
999 volume 3, pages 177–184.
999
- 1000 Thomas Kollar, Jayant Krishnamurthy, and Grant P
1001 Strimel. 2013. Toward interactive grounded lan-
1002 guage acquisition. In *Robotics: Science and systems*,
1003 volume 1, pages 721–732.
1004
- 1005 Stanislao Lauria, Guido Bugmann, Theocharis Kyri-
1006 acou, Johan Bos, and A Klein. 2001. Training
1007 personal robots using natural language instruction.
1008 *IEEE Intelligent systems*, 16(5):38–45.
1009
- 1010 Chen Liang, Jonathan Berant, Quoc Le, Kenneth D For-
1011 bus, and Ni Lao. 2016. Neural symbolic machines:
1012 Learning semantic parsers on freebase with weak su-
1013 pervision. *arXiv preprint arXiv:1611.00020*.
1014
- 1015 Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer,
1016 and Dieter Fox. 2013. Learning to parse natural lan-
1017 guage commands to a robot control system. In *Ex-
1018 perimental Robotics*, pages 403–415. Springer.
1019
- 1020 Arvind Neelakantan, Quoc V Le, Martin Abadi, An-
1021 drew McCallum, and Dario Amodei. 2016. Learn-
1022 ing a natural language interface with neural program-
1023 mer. *arXiv preprint arXiv:1611.08945*.
1024

- 1000 Junhyuk Oh, Valliappa Chockalingam, Satinder Singh,
1001 and Honglak Lee. 2016. Control of memory, active
1002 perception, and action in minecraft. *arXiv preprint*
1003 *arXiv:1605.09128*.
1004 Junhyuk Oh, Satinder Singh, Honglak Lee, and Push-
1005 meet Kohli. 2017. Zero-shot task generalization
1006 with multi-task deep reinforcement learning. *arXiv*
1007 *preprint arXiv:1706.05064*.
- 1008 Patti J Price. 1990. Evaluation of spoken language sys-
1009 tems: The atis domain. In *Speech and Natural Lan-*
1010 *guage: Proceedings of a Workshop Held at Hidden*
1011 *Valley, Pennsylvania, June 24-27, 1990*.
- 1012 Sebastian Riedel, Matko Bosnjak, and Tim
1013 Rocktäschel. 2016. Programming with a differen-
1014 tiable forth interpreter. *CoRR, abs/1605.06640*.
- 1015 Victor Sanh, Lysandre Debut, Julien Chaumond, and
1016 Thomas Wolf. 2019. Distilbert, a distilled version
1017 of BERT: smaller, faster, cheaper and lighter. *CoRR,*
abs/1910.01108.
- 1018 Tianmin Shu, Caiming Xiong, and Richard Socher.
1019 2017. Hierarchical and interpretable skill acqui-
1020 sition in multi-task reinforcement learning. *arXiv*
1021 *preprint arXiv:1712.07294*.
- 1022 Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-
1023 Yan Liu. 2019. MASS: masked sequence to se-
1024 quence pre-training for language generation. In *Pro-*
1025
1026 *Machine Learning, ICML 2019, 9-15 June 2019,*
Long Beach, California, USA, pages 5926–5936.
- 1027 Lappoon R Tang and Raymond J Mooney. 2001. Us-
1028 ing multiple clause constructors in inductive logic
1029 programming for semantic parsing. In *European*
1030 *Conference on Machine Learning*, pages 466–477.
Springer.
- 1031 Stefanie Tellex, Thomas Kollar, Steven Dickerson,
1032 Matthew R Walter, Ashis Gopal Banerjee, Seth
1033 Teller, and Nicholas Roy. 2011. Understanding nat-
1034 ural language commands for robotic navigation and
1035 mobile manipulation. In *Twenty-Fifth AAAI Confer-
1036 ence on Artificial Intelligence*.
- 1037 Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J
1038 Mankowitz, and Shie Mannor. 2017. A deep hierar-
1039 chical approach to lifelong learning in minecraft. In
1040 *AAAI*, volume 3, page 6.
- 1041 Jesse Thomason, Aishwarya Padmakumar, Jivko
1042 Sinapov, Nick Walker, Yuqian Jiang, Harel Yedid-
1043 sion, Justin Hart, Peter Stone, and Raymond J
1044 Mooney. 2019. Improving grounded natural lan-
1045 guage understanding through human-robot dialog.
arXiv preprint arXiv:1903.00122.
- 1046 Jesse Thomason, Shiqi Zhang, Raymond J Mooney,
1047 and Peter Stone. 2015. Learning to interpret nat-
1048 ural language commands through human-robot dialog.
In *Twenty-Fourth International Joint Conference on*
1049 *Artificial Intelligence*.
- 1050 Hiroto Udagawa, Tarun Narasimhan, and Shim-Young
1051 Lee. 2016. Fighting zombies in minecraft with deep
1052 reinforcement learning. Technical report, Technical
1053 report, Stanford University.
- 1054 Sida I Wang, Samuel Ginn, Percy Liang, and
1055 Christoper D Manning. 2017. Naturalizing a pro-
1056 gramming language via interactive learning. *arXiv*
1057 *preprint arXiv:1704.06956*.
- 1058 Sida I Wang, Percy Liang, and Christopher D Manning.
1059 2016. Learning language games through interaction.
arXiv preprint arXiv:1606.02447.
- 1060 Wenlu Wang, Yingtao Tian, Hongyu Xiong, Haixun
1061 Wang, and Wei-Shinn Ku. 2018. A transfer-
1062 learnable natural language interface for databases.
arXiv preprint arXiv:1809.02649.
- 1063 Yushi Wang, Jonathan Berant, and Percy Liang. 2015.
1064 Building a semantic parser overnight. In *Proceed-
1065 ings of the 53rd Annual Meeting of the Association
1066 for Computational Linguistics and the 7th Interna-
1067 tional Joint Conference on Natural Language Pro-
1068 cessing (Volume 1: Long Papers)*, volume 1, pages
1332–1342.
- 1069 Ziyu Yao, Yu Su, Huan Sun, and Wen-tau Yih. 2019.
1070 Model-based interactive semantic parsing: A uni-
1071 fied framework and A text-to-sql case study. *CoRR,*
abs/1910.05389.
- 1072 Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Tor-
1073 ralba, Pushmeet Kohli, and Josh Tenenbaum. 2018.
1074 Neural-symbolic vqa: Disentangling reasoning from
1075 vision and language understanding. In *Advances
1076 in Neural Information Processing Systems*, pages
1039–1050.
- 1077 Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-
1078 Wei Chang, and Jina Suh. 2016. The value of sem-
1079 antic parse labeling for knowledge base question
1080 answering. In *Proceedings of the 54th Annual Meet-
1081 ing of the Association for Computational Linguistics
1082 (Volume 2: Short Papers)*, pages 201–206.
- 1083 Victor Zhong, Caiming Xiong, and Richard Socher.
1084 2017. Seq2sql: Generating structured queries
1085 from natural language using reinforcement learning.
arXiv preprint arXiv:1709.00103.
- 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099

1100 **A Basic Data Cleanup** 1150
 1101

1102 We threw away all duplicate commands in the
 1103 dataset and only got annotations for unique com-
 1104 mands from each data source. 1154
 1105

1106 We performed post-processing on the text by
 1107 first inserting spaces between any special charac-
 1108 ter (brackets, “;”, “x”) followed by alphanumeric
 1109 character. For example “make a 5x5 hole” was
 1110 post-processed to “make a 5 x 5 hole” and “go
 1111 to (1,2,3)” to “go to (1 , 2 , 3)”. We then used
 1112 the tokenizer from spaCy <https://spacy.io/> to
 1113 tokenize every word in the sentence. 1161
 1114

1115 When constructing logical forms: we threw away
 1116 any keys with values : ‘None’ , ‘Other’ or ‘Not
 1117 Specified’ . Our tool allows workers to select these
 1118 options when annotating. We skipped stopwords
 1119 and articles like ‘a’ , ‘an’ etc when constructing
 1120 spans of children. We reordered the indices of
 1121 words in spans to always be from left to right (re-
 1122 gardless of which order the words were selected in
 1123 the sentence when annotating). 1170
 1124

1125 For commands annotated as “composite” (mean-
 1126 ing a command that requires multiple actions), we
 1127 set up another tool where we asked crowd-sourced
 1128 workers to split the composite command into indi-
 1129 vidual commands. Each of these commands were
 1130 then sent to our web-based tool described in 3.1.3
 1131 and the results were combined together under the
 1132 key: “action_sequence” by preserving the order.
 1133 So in the sentence: “jump twice and then come
 1134 to me”, we first have the sentence split into com-
 1135 mands: “jump twice” and “come to me” and then
 1136 combine their logical forms together under “ac-
 1137 tion_sequence” so we first have the “Dance” action
 1138 followed by “Move” action. This tool is described
 1139 in section B.4. 1184
 1140

1141

1142

1143

1144

1145

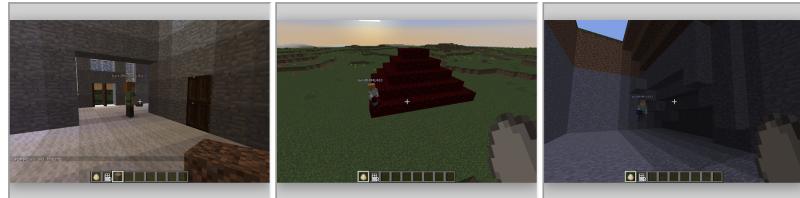
1146

1147

1148

1149

1200 Give instructions to a robot assistant in a creative game.



1201 1202 1203 1204 1205 1206 1207 1208 1209 1210 1211 1212 1213 1214 1215 1216 1217 1218 1219 1220 1221 1222 1223 1224 1225 1226 1227 1228 1229 1230 1231 1232 1233 1234 1235 1236 1237 1238 1239 1240 1241 1242 1243 1244 1245 1246 1247 1248 1249 1250 1251 1252 1253 1254 1255 1256 1257 1258 1259 1260 1261 1262 1263 1264 1265 1266 1267 1268 1269 1270 1271 1272 1273 1274 1275 1276 1277 1278 1279 1280 1281 1282 1283 1284 1285 1286 1287 1288 1289 1290 1291 1292 1293 1294 1295 1296 1297 1298 1299"/>

Imagine you are playing a creative game and have access to a robot assistant (as shown in the pictures above) that can help you get things done. Use your imagination and creativity to come up with three different instructions you'd like to give to the bot. Keep in mind that the bot is non-violent and can only do a defined set of things. Each command should only correspond to a single instruction or query (no "do this AND do that").

You can assume that the bot has the following capabilities:

- Move
- Build
- Destroy
- Tap
- Dig
- Copy
- Undo / Revert
- Fill
- Spawn
- Complete
- Answer
- Stop
- Resume

You can talk to the bot as you would to another human player to instruct it to perform actions mentioned above, ask questions, or give it information.

Please use your imagination to come up with things you'd like to say to the bot, given the capabilities. Be as creative as you can. The instructions should be related to the bot's capabilities, please give us as much variety as you can. Give us three unique instructions in the boxes below. Remember that the bot is non-violent.

Figure 7: The task instructions shown to crowd-sourced workers for the Image and text prompts task

B Crowd-sourced task and tools instructions

This section covers details of each crowd sourced task we've described in the paper along with screenshots of the web-based annotation tool described in 3.1.

B.1 Image and Text Prompts

In this task we showed a screenshot of the bot and environment to the crowd-sourced workers and asked them to give us free-form commands for the assistant. The instructions shown to workers are shown in 7.

B.2 Interactive Gameplay

In this task we had crowd-sourced workers play with our bot and interact with it using in-game chat. The instructions shown to workers are shown in 8.

B.3 Annotation tool

The web based annotation tool has two subparts: Tool a and Tool b.

B.3.1 Tool a

This tool is the first tool in the process of annotation and asks crowd-sourced workers to help determine the intent (dialogue_type or action_type) of the sentence and highlight other pieces of the text based on the choices they made for the intent. (For example: if the intent was “Build” they are asked to select words for the thing to be built and the location respectively.) We also provided helpful tooltips with examples at every step of the process.

The instructions shown to workers for Tool a are shown in figure 9 and step by step annotation process is shown in figure 10

B.3.2 Tool b

After we determine the intent from Tool a and get highlighted span of words for respective children of the intent, we use this tool. This is the second tool in the annotation process and asks crowd-sourced workers to help determine the fin-grained properties of specific entities of the action or dialogue. Note that we already got the words representing these, highlighted in B.3.1. For example : the words “ big bright house” are highlighted in the sentence “destroy the big bright house by the tree ” as an outcome of Tool a. The

1300	Welcome to the Minecraft Assistant project!	1350
1301	In this project, you will play Minecraft with a bot that is there to help you. Use the in-game chat to talk to the bot. Tell it where to go. Tell it what to do. Try to build something with its help.	1351
1302		1352
1303	For this project, you are asked to interact with the bot for 10 minutes . You may ask it to do anything you'd like. Explore what the bot can and cannot do.	1353
1304		1354
1305	The bot may ask you questions so it can learn over time. Please try to answer the questions it asks you.	1355
1306		1356
1307	When you are done, you will be asked to fill out a quick survey to help improve the bot.	1357
1308		1358
1309	To Proceed	1359
1310		1360
1311	1. Please make sure you have downloaded Minecraft and have a valid license. Follow the instructions here to change your version to 1.12	1361
1312	2. Press the green Launch button to access the loading screen.	1362
1313	3. After 1-2 minutes, you will see an IP address, like 123.123.123.123. Open Minecraft, navigate to Multiplayer > Direct Connect , enter the address, and click Join Server .	1363
1314	4. Play with the bot for 10 minutes . When you are done, exit Minecraft or press Escape > Disconnect .	1364
1315	5. Please fill out the short survey on the next page.	1365
1316	Please know that	1366
1317	Facebook AI Research will process the log of your game actions and in-game chats in accordance with our Data Policy. The log of your in-game actions and chats with the bot will be recorded for research purposes, and may be used by and/or shared with third parties in connection with this research. This may involve public disclosure of the log as part of a research paper or data set. We will take measures to remove any information that directly identifies you before doing so, but cannot guarantee that the logs will be completely anonymous. Do not send sensitive or personally identifiable information (for example, name, address, email, or phone number) in chats to the bot, and do not build structures with personally identifiable information (for example writing your name in blocks). Facebook's Community Standards apply and you may not use any racist, sexist, or otherwise offensive language. If you violate our policies you may be blocked.	1367
1318		1368
1319		1369
1320		1370
1321	Launch	1371
1322		1372
1323	Figure 8: The task instructions shown to crowd-sourced workers for the interactive game play	1373
1324		1374
1325		1375
1326	questionnaire changes dynamically based on the choices the workers make at every step of the tool. We provided helpful tooltips with examples at every step of the annotation process. Using the output of Tool a and Tool b, we can successfully construct the entire logical form for a given sentence.	1376
1327		1377
1328		1378
1329	The instructions shown to workers for Tool b are shown in figure 11 and step by step annotation	1379
1330	process for annotating properties of “location” in a “Move” action is shown in figure 12 and annotating	1380
1331	“reference_object” in “Destroy” action is shown in figure 13	1381
1332	B.4 Tool for composite commands	1382
1333	This tool is meant for “composite” commands (commands that include multiple actions) and asks the	1383
1334	users to split a command into multiple individual commands. The instruction for this are shown in figure	1384
1335	14. Once we get the split, we send out each command to annotation tool described in section B.3	1385
1336		1386
1337		1387
1338		1388
1339		1389
1340		1390
1341		1391
1342		1392
1343		1393
1344		1394
1345		1395
1346		1396
1347		1397
1348		1398
1349		1399

1400	1450
1401	1451
1402	1452
1403	1453
1404	1454
1405	1455
1406	1456
1407	1457
1408	1458
1409	1459
1410	1460
1411	1461
1412	1462
Instructions	
1413	1463
1414	1464
1415	1465
1416	1466
1417	1467
1418	1468
1419	1469
1420	1470
1421	1471
1422	1472
1423	1473
1424	1474
1425	1475
1426	1476
1427	1477
1428	1478
1429	1479
1430	1480
1431	1481
1432	1482
1433	1483
1434	1484
1435	1485
1436	1486
1437	1487
1438	1488
1439	1489
1440	1490
1441	1491
1442	1492
1443	1493
1444	1494
1445	1495
1446	1496
1447	1497
1448	1498
1449	1499

Figure 9: The task instructions shown to crowd-sourced workers for the annotation Tool a

1500	Command: build three sets of bookshelves in front of me .	1550
1501	What action is being requested? If multiple separate actions are being requested (e.g. "do X and then do Y"), select "Multiple separate actions" e.g. in 'Make few copies of the cube' it is : 'Build, make a copy or complete something'	1551
1502	<input type="radio"/> Build, make a copy or complete something <input type="radio"/> Move or walk somewhere <input type="radio"/> Spawn something (place an animal or creature in the game world) <input type="radio"/> Destroy, remove, or kill something <input type="radio"/> Dig <input type="radio"/> Fill something <input type="radio"/> Assign a description, name, or tag to an object <input type="radio"/> Answer a question <input type="radio"/> A movement where the path or step-sequence is more important than the destination <input type="radio"/> Stop an action <input type="radio"/> Resume an action <input type="radio"/> Undo or revert an action <input type="radio"/> Multiple separate actions <input type="radio"/> Another action not listed here <input type="radio"/> This sentence is not a command or request to do something	1552
1503		1553
1504		1554
1505		1555
1506		1556
1507		1557
1508		1558
1509	How many times should this action be performed?	1559
1510	<input type="radio"/> Just once, or not specified <input type="radio"/> Repeatedly, a specific number of times <input type="radio"/> Repeatedly, once for every object or for all objects <input type="radio"/> Repeated forever <input type="radio"/> Repeated until a certain condition is met	1560
1511		1561
1512		1562
1513	Command: build three sets of bookshelves in front of me .	1563
1514	What action is being requested? If multiple separate actions are being requested (e.g. "do X and then do Y"), select "Multiple separate actions" The sentence requests construction or making copies of some object	1564
1515	<input checked="" type="radio"/> Build, make a copy or complete something <input type="radio"/> Move or walk somewhere <input type="radio"/> Spawn something (place an animal or creature in the game world) <input type="radio"/> Destroy, remove, or kill something <input type="radio"/> Dig <input type="radio"/> Fill something <input type="radio"/> Assign a description, name, or tag to an object <input type="radio"/> Answer a question <input type="radio"/> A movement where the path or step-sequence is more important than the destination <input type="radio"/> Stop an action <input type="radio"/> Resume an action <input type="radio"/> Undo or revert an action <input type="radio"/> Multiple separate actions <input type="radio"/> Another action not listed here <input type="radio"/> This sentence is not a command or request to do something	1565
1516		1566
1517		1567
1518		1568
1519		1569
1520		1570
1521		1571
1522	Is this exact copy or duplicate of something that already exists?	1572
1523	<input type="radio"/> Yes <input checked="" type="radio"/> No	1573
1524	Is the assistant being asked to...	1574
1525		1575
1526	Command: build three sets of bookshelves in front of me .	1576
1527	Is this exact copy or duplicate of something that already exists?	1577
1528	<input type="radio"/> Yes <input checked="" type="radio"/> No	1578
1529	Is the assistant being asked to...	1579
1530	<input checked="" type="radio"/> Build a specific object or objects from scratch <input type="radio"/> Help complete or finish already existing object(s) e.g. in 'construct two big wooden houses in front of the tower' select 'big wooden houses'	1580
1531	Select words specifying what needs to be built build three sets of bookshelves in front of me .	1581
1532	Select words specifying where the thing should be built Click and select all words if specified build three sets of bookshelves in front of me .	1582
1533		1583
1534		1584
1535		1585
1536	Command: build three sets of bookshelves in front of me .	1586
1537	How many times should this action be performed?	1587
1538	<input type="radio"/> Just once, or not specified <input checked="" type="radio"/> Repeatedly, a specific number of times The action needs to be repeated a fixed number of times <input type="radio"/> Repeatedly, once for every object or for all objects times <input type="radio"/> Repeated forever <input type="radio"/> Repeated until a certain condition is met	1588
1539		1589
1540	How many times? Select all words build three sets of bookshelves in front of me .	1590
1541	In which direction should the action be repeated?	1591
1542	<input checked="" type="radio"/> Not specified <input type="radio"/> Forward <input type="radio"/> Backward <input type="radio"/> Left <input type="radio"/> Right <input type="radio"/> Up <input type="radio"/> Down <input type="radio"/> Around <input type="radio"/> Other	1592
1543		1593
1544		1594
1545		1595
1546		1596
1547	<input type="button" value="Submit"/>	1597
1548	Figure 10: The step by step screenshot of annotations process for the command: "build three sets of bookshelves in front of me ." in Tool a	1598
1549		1599

1600	1650
1601	1651
1602	1652
1603	1653
1604	1654
1605	1655
1606	1656
1607	1657
1608	1658
1609	1659
1610	1660
1611	1661
1612	1662
1613	1663
1614	1664
Instructions	
1615	1665
1616	1666
1617	1667
1618	1668
1619	1669
1620	1670
1621	1671
1622	1672
1623	1673
1624	1674
1625	1675
1626	1676
1627	1677
1628	1678
1629	1679
1630	1680
1631	1681
1632	1682
1633	1683
1634	1684
1635	1685
1636	1686
1637	1687
1638	1688
1639	1689
1640	1690
1641	1691
1642	1692
1643	1693
1644	1694
1645	1695
1646	1696
1647	1697
1648	1698
1649	1699

Figure 11: The task instructions shown to crowd-sourced workers for the annotation Tool b

1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780

Command: go 5 steps in front of that

Please specify details of where the assistant should move.

Where should the assistant move to

Not specified
 The location is represented using an indefinite noun like 'there' or 'over here'
 Exact numerical coordinates are given
 Where the speaker is looking
 Somewhere relative to where the speaker is looking e.g. 'in front of where I am looking'
 Where the speaker is standing
 Somewhere relative to where the speaker is standing
 Where the assistant is standing
 Somewhere relative to where the assistant is standing
 Somewhere relative to (or exactly at) another object(s) / area(s)
 Other

If a number of steps is specified, how many?

Submit

Command: go 5 steps in front of that

Somewhere relative to where the assistant is standing
 Somewhere relative to (or exactly at) another object(s) / area(s)
 Other

Where (which direction) in relation to the other object(s)?

In front
 Behind
 Away from
 Inside
 Outside
 Between two object(s) / area(s)
 Nearby or close to
 Around
 Exactly at
 Other

Click on all words specifying the object / area in front of which the location is given

e.g. 'to the right of this', 'near that', 'behind these', 'next to those', 'underneath it' etc

Are there indefinite nouns or pronouns specifying the relative object?
 Yes
 No

Command: go 5 steps in front of that

Away from
 Inside
 Outside
 Between two object(s) / area(s)
 Nearby or close to
 Around
 Exactly at
 Other

Click on all words specifying the object / area in front of which the location is given

Are there indefinite nouns or pronouns specifying the relative object?
 Yes
 No

If a number of steps is specified, how many?

Submit

Figure 12: The step by step screenshot of annotating properties of highlighted words for “location” in a “Move” action.

1800		1850
1801		1851
1802		1852
1803		1853
1804		1854
1805		1855
1806		1856
1807		1857
1808		1858
1809		1859
1810		1860
1811		1861
1812		1862
1813	Command: destroy the big bright house by the tree	1863
1814	Please specify details of the thing that needs to be destroyed. Click on all mentioned properties of the object in highlighted text.	1864
1815	<input type="radio"/> Name <input type="radio"/> There are words or pronouns that refer to the object to be destroyed (e.g. 'this', 'that', 'these', 'those', 'it' etc) <input type="radio"/> The building material <input type="radio"/> Colour <input type="radio"/> Abstract/non-numeric size (e.g. 'big', 'small', etc.) <input type="radio"/> Height <input type="radio"/> Length <input type="radio"/> Width <input type="radio"/> Depth <input checked="" type="radio"/> Some other property not mentioned above	1865
1816	<small>Select this if any property not explicitly mentioned above is given</small>	1866
1817		1867
1818		1868
1819		1869
1820		1870
1821		1871
1822	Command: destroy the big bright house by the tree	1872
1823	Please specify details of the thing that needs to be destroyed. Click on all mentioned properties of the object in highlighted text.	1873
1824	<input checked="" type="radio"/> Name <input type="radio"/> There are words or pronouns that refer to the object to be destroyed (e.g. 'this', 'that', 'these', 'those', 'it' etc) <input type="radio"/> The building material <input type="radio"/> Colour <input checked="" type="radio"/> Abstract/non-numeric size (e.g. 'big', 'small', etc.) <input type="radio"/> Height <input type="radio"/> Length <input type="radio"/> Width <input type="radio"/> Depth <input checked="" type="radio"/> Some other property not mentioned above	1874
1825	<small>What is the name of the object that should be destroyed?</small> destroy the big bright house by the tree	1875
1826		1876
1827		1877
1828		1878
1829		1879
1830	<small>What is the size?</small> destroy the big bright house by the tree	1880
1831		1881
1832		1882
1833	<small>Select all words for this property</small> destroy the big bright house by the tree	1883
1834		1884
1835	Figure 13: The step by step screenshot of annotating properties of highlighted words for“reference_object” in a “Destroy” action.	1885
1836		1886
1837		1887
1838		1888
1839		1889
1840		1890
1841		1891
1842		1892
1843		1893
1844		1894
1845		1895
1846		1896
1847		1897
1848		1898
1849		1899

1900	1950
1901	1951
1902	1952
1903	1953
1904	1954
1905	1955
1906	1956
1907	1957
1908	1958
1909	1959
1910	1960
1911	1961
1912	1962
1913	1963
1914	1964
1915	1965
1916	1966
1917	1967
1918	1968
1919	1969
1920	1970
1921	1971
1922	1972
1923	1973
1924	1974
1925	1975
1926	1976
1927	1977
1928	1978
1929	1979
1930	1980
1931	1981
1932	1982
1933	1983
1934	1984
1935	1985
1936	1986
1937	1987
1938	1988
1939	1989
1940	1990
1941	1991
1942	1992
1943	1993
1944	1994
1945	1995
1946	1996
1947	1997
1948	1998
1949	1999

Instructions

Split a composite command into individuals.

Please help us split a command into individual single commands. The command shown to you here is given to an AI assistant to help out a player in the game of Minecraft. You will be show a command that possibly implies a sequence or list of single commands and your task is to give us single complete actions that are intended by the command shown to you.

Few valid examples below:

For "**hey bot please build a house and a cube**" the answer is the following:

- "hey bot please build a house" and
- "hey bot please build a cube"

For "**build a castle and then come back here**" the answer is the following:

- "build a castle" and
- "come back here"

For "**destroy the roof and build a stone ceiling in its place**" the answer is the following:

- "destroy the roof" and
- "build a stone ceiling in its place"

For "**move to the door and open it**" the answer is the following:

- "move to the door" and
- "open the door"

For "**i want you to undo the last two spawns and try again with new spawns**"

- "undo the last two spawns" and
- "do a new spawn"

Note that: "**do a new spawn**" is a rewrite of "**and try again with new spawns**" to make that sub-command clear when seen in isolation.

Note that:

1. Some commands might have more than two splits. We've given you two more optional boxes.
2. Make sure that the commands you enter in text boxes are single and complete sentences by their own.
3. You might need to rewrite some commands when you split them, to make them clear in isolation.

Figure 14: The task instructions shown to crowd-sourced workers for splitting composite commands

2000 C Action Tree structure

2001
 2002 This section describes the details of logical form of each action. We support three dialogue types:
 2003 HUMAN_GIVE_COMMAND, GET_MEMORY and PUT_MEMORY. The logical form for actions has
 2004 been pictorially represented in figures: [1](#) and [2](#)

2005 We support the following actions in our dataset : Build, Copy, Dance, Spawn, Resume, Fill, Destroy,
 2006 Move, Undo, Stop, Dig and FreeBuild. A lot of the actions use “location” and “reference_object” as
 2007 children in their logical forms. To make the logical forms more presentable, we have shown the detailed
 2008 representation of a “reference_object” (reused in action trees using the variable: “REF_OBJECT”) in
 2009 figure [15](#) and the representation of “location” (reused in action trees using the variable: “LOCATION”) in
 2010 figure [16](#). The representations of actions refer to these variable names in their trees.

```
2011 REF_OBJECT:
2012
2013   The recursion depth of REF_OBJECT in LOCATION here was never greater than 1 in the data.
2014   so a REF_OBJECT can have a LOCATION that has a REF_OBJECT that has a LOCATION (and the final location will be one of :
2015     COORDINATES / AGENT_POS / SPEAKER_POS / SPEAKER_LOOK).
2016
2017   "reference_object" : {
2018     "repeat" : {
2019       "repeat_key" : 'FOR' / 'ALL'
2020       "repeat_count" : span,
2021       "repeat_dir" : 'LEFT' / 'RIGHT' / 'UP' / 'DOWN' / 'FRONT' / 'BACK' / 'AROUND'
2022     }
2023     "has_name" : span,
2024     "has_size" : span,
2025     "has_colour" : span,
2026     "has_tag" : span,
2027     "has_length" : span,
2028     "has_width" : span,
2029     "has_height" : span,
2030     "contains_coreference" : "yes",
2031     LOCATION
2032   }
```

2033 Figure 15: Logical form of a reference_object child

```
2034
2035 LOCATION:
2036
2037   "location" : {
2038     "location_type" : COORDINATES / REFERENCE_OBJECT / AGENT_POS / SPEAKER_POS / SPEAKER_LOOK,
2039     "steps" : span,
2040     "contains_coreference" : "yes",
2041     "relative_direction" : 'LEFT' / 'RIGHT' / 'UP' / 'DOWN' / 'FRONT' / 'BACK' /
2042       'AWAY' / 'INSIDE' / 'NEAR' / 'OUTSIDE' / 'BETWEEN',
2043     "coordinates" : span, (present if "location_type" == 'COORDINATES')
2044     REF_OBJECT (present if "location_type" == 'REFERENCE_OBJECT')
2045   },
```

2046 Figure 16: Logical form of a location child

2047 The detailed action tree for each action and dialogue type has been presented in the following subsections. Figure [17](#) shows an example for a BUILD action.

2048 C.1 Build Action

2049 This is the action to Build a schematic at an optional location. The Build logical form is shown in [18](#).

2050 C.2 Copy Action

2051 This is the action to copy a block object to an optional location. The copy action is represented as a
 2052 “Build” with an optional “reference object”. The logical form is shown in [19](#).

2053 C.3 Spawn Action

2054 This action indicates that the specified object should be spawned in the environment. The logical form is
 2055 shown in: [20](#)

```

2100      0   1   2   3   4   5   6
2101 "Make three oak wood houses to the
2102     7   8   9   10  11  12
2103 left of the dark grey church."
2104
2105 {"dialogue_type" : "HUMAN_GIVE_COMMAND",
2106   "action_sequence" : [
2107     {
2108       "action_type" : "BUILD",
2109       "schematic": {
2110         "has_block_type": [0, [2, 3]],
2111         "has_name": [0, [4, 4]],
2112         "repeat": {
2113           "repeat_key": "FOR",
2114           "repeat_count": [1, 1]
2115         },
2116         "location": {
2117           "relative_direction": "LEFT",
2118           "location_type": "REFERENCE_OBJECT",
2119           "reference_object": {
2120             "has_colour_": [0, [10, 11]],
2121             "has_name_": [0, [12, 12]] }
2122       } },
2123     } ] }

```

Figure 17: An example logical form. The spans are indexed as : [sentence_number, [starting_word_index, ending_word_index]]. sentence_number is 0 for the most recent sentence spoken in a dialogue and is 0 in our dataset since we support one-turn dialogues as of now.

C.4 Fill Action

This action states that a hole / negative shape at an optional location needs to be filled up. The logical form is explained in : 21

C.5 Destroy Action

This action indicates the intent to destroy a block object at an optional location. The logical form is shown in: 22

Destroy action can have one of the following as the child:

- reference object
- nothing

C.6 Move Action

This action states that the agent should move to the specified location, the corresponding logical form is in: 23

Move action can have one of the following as its child:

- location
- stop condition (stop moving when a condition is met)
- location and stop condition
- neither

C.7 Dig Action

This action represents the intent to dig a hole / negative shape of optional dimensions at an optional location. The logical form is in 24

C.8 Dance Action

This action represents that the agent performs a movement of a certain kind. Note that this action is different than a Move action in that the path or step-sequence here is more important than the destination. The logical form is shown in 25

```

2200 { "dialogue_type": "HUMAN_GIVE_COMMAND",
2201   "action_sequence" : [
2202     { "action_type" : 'BUILD',
2203       LOCATION,
2204       "schematic" : {
2205         "repeat" : {
2206           "repeat_key" : 'FOR'
2207           "repeat_count" : span,
2208           "repeat_dir": 'LEFT' / 'RIGHT' / 'UP' / 'DOWN' / 'FRONT' / 'BACK' / 'AROUND' / 'SURROUND'
2209         }
2210       "has_block_type" : span,
2211       "has_name": span,
2212       "has_size" : span,
2213       "has_orientation" : span,
2214       "has_thickness" : span,
2215       "has_colour" : span,
2216       "has_height" : span,
2217       "has_length" : span,
2218       "has_radius" : span,
2219       "has_slope" : span,
2220       "has_width" : span,
2221       "has_base" : span,
2222       "has_distance" : span,
2223     },
2224     "repeat" : {
2225       "repeat_key" : 'FOR'
2226       "repeat_count" : span,
2227       "repeat_dir": 'LEFT' / 'RIGHT' / 'UP' / 'DOWN' / 'FRONT' / 'BACK' / 'AROUND' / 'SURROUND'
2228     } } ] }

```

Figure 18: Details of logical form for Build

```

2229 { "dialogue_type": "HUMAN_GIVE_COMMAND",
2230   "action_sequence" : [
2231     { "action_type" : 'BUILD',
2232       LOCATION,
2233       REF_OBJ,
2234       "repeat" : {
2235         "repeat_key" : 'FOR'
2236         "repeat_count" : span,
2237         "repeat_dir": 'LEFT' / 'RIGHT' / 'UP' / 'DOWN' / 'FRONT' / 'BACK' / 'AROUND'
2238       } } ] }

```

Figure 19: Details of logical form for Copy

C.9 FreeBuild Action

This action represents that the agent should complete an already existing half-finished block object, using its mental model. The logical form is explained in: [26](#)

FreeBuild action can have one of the following as its child:

- reference object only
- reference object and location

C.10 Undo Action

This action states the intent to revert the specified action, if any. The logical form is in [27](#). Undo action can have one of the following as its child:

- target_action_type
- nothing (meaning : undo the last action)

C.11 Stop Action

This action indicates stop and the logical form is shown in [28](#)

```

2300     { "dialogue_type": "HUMAN_GIVE_COMMAND",
2301         "action_sequence" : [
2302             { "action_type" : 'SPAWN'
2303                 REF_OBJ,
2304                 LOCATION } ] }

```

Figure 20: Details of logical form for Spawn action

```

2306     { "dialogue_type": "HUMAN_GIVE_COMMAND",
2307         "action_sequence" : [
2308             { "action_type" : 'FILL',
2309                 "has_block_type" : span,
2310                 REF_OBJECT } ] }

```

Figure 21: Details of logical form for Fill

C.12 Resume Action

This action indicates that the previous action should be resumed, the logical form is shown in: [29](#)

C.13 Get Memory Dialogue type

This dialogue type represents the agent answering a question about the environment. This is similar to the setup in Visual Question Answering. The logical form is represented in: [30](#)

Get Memory dialogue has the following as its children: filters, answer type and tag name. This dialogue type represents the type of expected answer : counting, querying a specific attribute or querying everything ("what is the size of X" vs "what is X")

C.14 Put Memory Dialogue

This dialogue type represents that a reference object should be tagged with the given tag and the logical form is shown in: [31](#)

C.15 Noop Dialogue

This dialogue type indicates no operation should be performed, the logical form is shown in : [32](#)

```

2400             { "dialogue_type": "HUMAN_GIVE_COMMAND",
2401               "action_sequence" : [
2402                 { "action_type" : 'DESTROY',
2403                   REF_OBJECT,
2404                 } ] }

```

Figure 22: Details of logical form Destroy

```

2405
2406
2407
2408 { "dialogue_type": "HUMAN_GIVE_COMMAND",
2409   "action_sequence" : [
2410     { "action_type" : 'MOVE',
2411       LOCATION,
2412       "stop_condition" : {
2413         "condition_type" : 'ADJACENT_TO_BLOCK_TYPE' / 'NEVER',
2414         "block_type" : span,
2415         "condition_span" : span,
2416       },
2417       "repeat" : {
2418         "repeat_key" : 'FOR',
2419         "repeat_count" : span,
2420         "repeat_dir": 'LEFT' / 'RIGHT' / 'UP' / 'DOWN' / 'FRONT' / 'BACK' / 'AROUND'
2421       }
2422     }
2423   }
2424 }
```

Figure 23: Details of logical form for Move action

```

2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449 { "dialogue_type": "HUMAN_GIVE_COMMAND",
2450   "action_sequence" : [
2451     { "action_type" : 'DIG',
2452       LOCATION,
2453       "schematic" : {
2454         "repeat" : {
2455           "repeat_key" : 'FOR'
2456           "repeat_count" : span,
2457           "repeat_dir": 'LEFT' / 'RIGHT' / 'UP' / 'DOWN' / 'FRONT' / 'BACK' / 'AROUND'
2458         }
2459         "has_size" : span,
2460         "has_length" : span,
2461         "has_depth" : span,
2462         "has_width" : span,
2463       },
2464       "stop_condition" : {
2465         "condition_type" : 'ADJACENT_TO_BLOCK_TYPE' / 'NEVER',
2466         "block_type" : span
2467       }
2468     }
2469   }
2470 }
```

Figure 24: Details of logical form for Dig action

```

2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499 { "dialogue_type": "HUMAN_GIVE_COMMAND",
2500   "action_sequence" : [
2501     { "action_type" : 'DANCE',
2502       LOCATION,
2503       "stop_condition" : {
2504         "condition_type" : NEVER,
2505       },
2506       "repeat" : {
2507         "repeat_key" : 'FOR',
2508         "repeat_count" : span,
2509         "repeat_dir": 'LEFT' / 'RIGHT' / 'UP' / 'DOWN' / 'FRONT' / 'BACK' / 'AROUND'
2510       }
2511     }
2512   }
2513 }
```

Figure 25: Details of logical form for Dance action

```

2500             { "dialogue_type": "HUMAN_GIVE_COMMAND",
2501                 "action_sequence" : [
2502                     { "action_type" : 'FREEBUILD',
2503                         REF_OBJECT,
2504                         LOCATION } ] }

```

Figure 26: Details of logical form for FreeBuild action

```

2505
2506             { "dialogue_type": "HUMAN_GIVE_COMMAND",
2507                 "action_sequence" : [
2508                     { "action_type" : 'UNDO',
2509                         "target_action_type" : span
2510                     }
2511                 ]

```

Figure 27: Details of logical form for Undo action

```

2512             { "dialogue_type": "HUMAN_GIVE_COMMAND",
2513                 "action_sequence" : [
2514                     { "action_type" : 'STOP',
2515                         "target_action_type": span
2516                     }
2517                 ]

```

Figure 28: Details of logical form for Stop action

```

2518             { "dialogue_type": "HUMAN_GIVE_COMMAND",
2519                 "action_sequence" : [
2520                     { "action_type" : 'RESUME',
2521                         "target_action_type": span
2522                     }
2523                 ]

```

Figure 29: Details of logical form for Resume action

```

2523 {
2524     "dialogue_type": "GET_MEMORY",
2525     "filters": {
2526         "temporal": CURRENT,
2527         "type": "ACTION" / "AGENT" / "REFERENCE_OBJECT",
2528         "action_type": BUILD / DESTROY / DIG / FILL / SPAWN / MOVE
2529         REF_OBJECT
2530     },
2531     "answer_type": "TAG" / "EXISTS" ,
2532     "tag_name" : 'has_name' / 'has_size' / 'has_colour' / 'action_name' /
2533             'action_reference_object_name' / 'move_target' / 'location'
2534 }

```

Figure 30: Details of logical form for Get Memory Dialogue

```

2534 {
2535     "dialogue_type": "PUT_MEMORY",
2536     "filters": {
2537         REF_OBJECT },
2538     "upsert" : {
2539         "memory_data": {
2540             "memory_type": "REWARD" / "TRIPLE",
2541             "reward_value": "POSITIVE" / "NEGATIVE",
2542             "has_tag" : span,
2543             "has_colour": span,
2544             "has_size": span
2545         } } }

```

Figure 31: Details of logical form for Put Memory Dialogue

```

2546             { "dialogue_type": "NOOP"}
```

Figure 32: Details of logical form for Noop Dialogue

D Crowd-sourced task and tools instructions

Some examples from prompts data:

```

2600
2601
2602
2603
2604 bot move the tree to the left side of the house
2605 {'action_sequence': [{}{'action_type': 'OTHERACTION',
2606     'location': {}{'location_type': 'REFERENCE_OBJECT',
2607         'reference_object': {}{'has_name': [0,
2608             [10, 10]]},
2609             'relative_direction': 'LEFT'},
2610         'reference_object': {}{'has_name': [0, [3, 3]]}}],
2611     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2612
2613 dig a hole next to that house
2614 {'action_sequence': [{}{'action_type': 'DIG',
2615     'location': {}{'location_type': 'REFERENCE_OBJECT',
2616         'reference_object': {}{'contains_coreference': 'yes',
2617             'has_name': [0,
2618                 [6, 6]]},
2619             'relative_direction': 'NEAR'},
2620         'schematic': {}{'has_name': [0, [2, 2]]}}},
2621     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2622
2623 how about you copy the crops i planted to fill this whole plain
2624 {'action_sequence': [{}{'action_type': 'BUILD',
2625     'reference_object': {}{'has_name': [0, [5, 5]],
2626         'has_tag': [0, [6, 7]]},
2627     'repeat': {}{'stop_condition': {}{'condition_span': [0,
2628                 [9,
2629                     [12]]}}}},
2630     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2631
2632 make sure i spawn on top of the pyramid each time
2633 {'action_sequence': [{}{'action_type': 'OTHERACTION',
2634     'location': {}{'location_type': 'REFERENCE_OBJECT',
2635         'reference_object': {}{'has_name': [0,
2636             [8, 8]]},
2637             'relative_direction': 'UP'},
2638         'reference_object': {}{'has_name': [0, [2, 2]]},
2639         'repeat': {}{'stop_condition': {}{'condition_type': 'NEVER'}}}],
2640     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2641
2642 complete the structure 10 meters west from your position
2643 {'action_sequence': [{}{'action_type': 'FREEBUILD',
2644     'reference_object': {}{'has_name': [0, [2, 2]],
2645         'location': {}{'location_type': 'AGENT_POS',
2646             'relative_direction': 'LEFT',
2647                 'steps': [0,
2648                     [3, 3]]}}},
2649     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2650
2651 destroy the structure that is blocking the view of the landscape
2652 {'action_sequence': [{}{'action_type': 'DESTROY',
2653     'reference_object': {}{'has_name': [0, [2, 2]],
2654         'has_tag': [0, [5, 10]]}}},
2655     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2656
2657 complete the project that i am working on by building more devices
2658 {'action_sequence': [{}{'action_type': 'FREEBUILD',
2659     'reference_object': {}{'has_name': [0, [2, 2]],
2660         'has_tag': [0, [4, 7]]}}},
2661     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2662
2663 show me how to dance
2664 {'action_sequence': [{}{'action_type': 'DANCE'}],
2665     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2666
2667 please build a garden
2668 {'action_sequence': [{}{'action_type': 'BUILD',
2669     'location': {}{'location_type': 'REFERENCE_OBJECT',
2670         'reference_object': {}{'has_name': [0,
2671             [10, 10]]},
2672             'relative_direction': 'LEFT'},
2673         'reference_object': {}{'has_name': [0, [3, 3]]}}},
2674     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699

```

```

2700             'schematic': {'has_name': [0, [3, 3]]]}], 2750
2701     'dialogue_type': 'HUMAN_GIVE_COMMAND' 2751

2702     fill the small pond with sand 2752
2703     {'action_sequence': [{ 2753
2704         'action_type': 'FILL', 2754
2705         'has_block_type': [0, [5, 5]], 2755
2706         'reference_object': {'has_name': [0, [3, 3]], 2756
2707             'has_size': [0, [2, 2]]]}], 2756
2708     'dialogue_type': 'HUMAN_GIVE_COMMAND' 2757

2709     move north for 5 minutes 2758
2710     {'action_sequence': [ 2759
2711         {'action_type': 'MOVE', 2760
2712             'location': {'location_type': 'AGENT_POS', 2761
2713                 'relative_direction': 'FRONT'}, 2761
2714             'repeat': {'stop_condition': {'condition_span': [0, 2762
2715                 [3, 4]]}}}], 2762
2716     'dialogue_type': 'HUMAN_GIVE_COMMAND' 2763

2717     dig a hole next to the sidewalk of the school 2764
2718     {'action_sequence': [ 2765
2719         {'action_type': 'DIG', 2766
2720             'location': {'location_type': 'REFERENCE_OBJECT', 2767
2721                 'reference_object': {'has_name': [0, 2768
2722                     [6, 9]]}, 2769
2723                 'relative_direction': 'NEAR'}, 2769
2724             'schematic': {'has_name': [0, [2, 2]]]}], 2770
2725     'dialogue_type': 'HUMAN_GIVE_COMMAND' 2770

2726     move to the right until you ca n't anymore 2770
2727     {'action_sequence': [ 2771
2728         {'action_type': 'MOVE', 2772
2729             'location': {'location_type': 'SPEAKER_POS', 2773
2730                 'relative_direction': 'RIGHT'}, 2773
2731             'repeat': {'stop_condition': {'condition_span': [0, 2774
2732                 [4, 8]]}}}], 2774
2733     'dialogue_type': 'HUMAN_GIVE_COMMAND' 2775

2734     move up the hill 2776
2735     {'action_sequence': [ 2777
2736         {'action_type': 'MOVE', 2778
2737             'location': {'location_type': 'REFERENCE_OBJECT', 2779
2738                 'reference_object': {'has_name': [0, 2780
2739                     [3, 3]]}, 2781
2740                 'relative_direction': 'UP'}]}], 2781
2741     'dialogue_type': 'HUMAN_GIVE_COMMAND' 2782

2742     build a bridge over the lava 2782
2743     {'action_sequence': [ 2783
2744         {'action_type': 'BUILD', 2784
2745             'location': {'location_type': 'REFERENCE_OBJECT', 2785
2746                 'reference_object': {'has_name': [0, 2786
2747                     [5, 5]]}, 2787
2748                 'relative_direction': 'UP'}, 2787
2749             'schematic': {'has_name': [0, [2, 2]]]}], 2788
2750     'dialogue_type': 'HUMAN_GIVE_COMMAND' 2788

2751     this pyramid is 5 platforms tall 2788
2752     {'dialogue_type': 'NOOP'} 2789

2753     spawn 30 cows and build a 15 by 15 fence 2790
2754     {'action_sequence': [ 2791
2755         {'action_type': 'SPAWN', 2792
2756             'reference_object': {'has_name': [0, [2, 2]]}, 2793
2757             'repeat': {'repeat_count': [0, [1, 1]], 2794
2758                 'repeat_key': 'FOR'}}, 2794
2759             {'action_type': 'BUILD', 2795
2760                 'schematic': {'has_height': [0, [2, 2]], 2795
2761                     'has_name': [0, [5, 5]]]}], 2796
2762     'dialogue_type': 'HUMAN_GIVE_COMMAND' 2796

2763     move three feet forward and stop 2797
2764     {'action_sequence': [ 2798
2765         {'action_type': 'MOVE', 2799
2766             'location': {'location_type': 'AGENT_POS', 2799
2767                 'relative_direction': 'FRONT'}]}], 2799

```

```

2800                 'relative_direction': 'FRONT',
2801                 'steps': [0, [1, 2]]}]], 2850
2802             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2851
2803
2804         destroy the building that 's in front of you 2853
2805             {'action_sequence': [{{'action_type': 'DESTROY', 2854
2806                 'reference_object': {'has_name': [0, [2, 2]], 2855
2807                     'location': {'location_type': 'AGENT_POS', 2856
2808                         'relative_direction': 'FRONT'}}}], 2857
2809             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2858
2810
2811         tag the horse armor 2858
2812             {'dialogue_type': 'PUT_MEMORY', 2859
2813                 'filters': {'reference_object': {'has_name': [0, [2, 3]]}}}] 2860
2814
2815         bot build it to fit into the open frame 2861
2816             {'action_sequence': [{{'action_type': 'BUILD', 2862
2817                 'schematic': {'has_name': [0, [2, 2]], 2863
2818                     'has_tag': [0, [4, 8]]}}], 2864
2819             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2865
2820
2821         destroy the hut near the big tree 2866
2822             {'action_sequence': [{{'action_type': 'DESTROY', 2867
2823                 'reference_object': {'has_name': [0, [2, 2]]}}}], 2868
2824             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2869
2825
2826         move the rabbit into the box 2870
2827             {'action_sequence': [{{'action_type': 'OTHERACTION', 2871
2828                 'location': {'location_type': 'REFERENCE_OBJECT', 2872
2829                     'reference_object': {'has_name': [0, 2873
2830                         [5, 5]]}, 2874
2831                         'relative_direction': 'INSIDE'}, 2875
2832                     'reference_object': {'has_name': [0, [2, 2]]}}}], 2876
2833             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2877
2834
2835         fill the entire tub with pepsi 2878
2836             {'action_sequence': [{{'action_type': 'FILL', 2879
2837                 'has_block_type': [0, [5, 5]], 2880
2838                     'reference_object': {'has_name': [0, [3, 3]]}}}], 2881
2839             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2882
2840
2841         stop digging 2883
2842             {'action_sequence': [{{'action_type': 'STOP', 2884
2843                 'target_action_type': [0, [1, 1]]}}], 2885
2844             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2886
2845
2846         destroy the box 2887
2847             {'action_sequence': [{{'action_type': 'DESTROY', 2888
2848                 'reference_object': {'has_name': [0, [2, 2]]}}}], 2889
2849             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2890
2850
2851         let 's resume our mission of traveling over that treacherous mountain pass 2891
2852             {'action_sequence': [{{'action_type': 'RESUME', 2892
2853                 'target_action_type': [0, [3, 11]]}}], 2894
2854             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2895
2855
2856         build a house with a porch next to the pyramid 2896
2857             {'action_sequence': [{{'action_type': 'BUILD', 2897
2858                 'location': {'location_type': 'REFERENCE_OBJECT', 2898
2859                     'reference_object': {'has_name': [0, 2860
2861                         [9, 9]]}, 2862
2862                         'relative_direction': 'NEAR'}, 2863
2863                     'schematic': {'has_name': [0, [2, 2]], 2864
2864                         'has_tag': [0, [3, 5]]}}], 2865
2865             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2866
2866
2867         build stairs in the corner 2867
2868             {'action_sequence': [{{'action_type': 'BUILD', 2868
2869                 'location': {'location_type': 'REFERENCE_OBJECT', 2869
2870                     'reference_object': {'has_name': [0, 2871
2871                         [9, 9]]}}}], 2872
2872             'dialogue_type': 'HUMAN_GIVE_COMMAND'} 2873

```

```

2900                               [4, 4]]}},           2950
2901             'schematic': {'has_name': [0, [1, 1]]}}],   2951
2902       'dialogue_type': 'HUMAN_GIVE_COMMAND' }          2952

2903 spawn milk                                         2953
2904   {'action_sequence': [{ 'action_type': 'SPAWN',      2954
2905     'reference_object': {'has_name': [0, [1, 1]]}}],   2955
2906   'dialogue_type': 'HUMAN_GIVE_COMMAND' }

2907 build a wall to divide the largest room in the house 2956
2908   {'action_sequence': [{ 'action_type': 'BUILD',        2957
2909     'location': {'location_type': 'REFERENCE_OBJECT', 2958
2910       'reference_object': {'has_name': [0, [6, 10]]},    2959
2911       'relative_direction': 'INSIDE'},                 2960
2912     'schematic': {'has_name': [0, [2, 2]]}}],           2961
2913   'dialogue_type': 'HUMAN_GIVE_COMMAND' }

2914 build foundation                                     2963
2915   {'action_sequence': [{ 'action_type': 'BUILD',        2964
2916     'schematic': {'has_name': [0, [1, 1]]}}],           2965
2917   'dialogue_type': 'HUMAN_GIVE_COMMAND' }

2918 please change the barn to a shop                   2966
2919   {'action_sequence': [{ 'action_type': 'OTHERACTION', 2967
2920     'reference_object': {'has_name': [0, [3, 3]]}}],       2968
2921   'dialogue_type': 'HUMAN_GIVE_COMMAND' }

2922 copy the loaf of bread 100 times for distribution to the assembled army in front of 2970
2923   you                                         2971
2924   {'action_sequence': [{ 'action_type': 'BUILD',        2972
2925     'reference_object': {'has_name': [0, [2, 4]]},       2973
2926     'repeat': {'repeat_count': [0, [5, 5]],            2974
2927       'repeat_key': 'FOR'}}],                         2975
2928   'dialogue_type': 'HUMAN_GIVE_COMMAND' }

2929 spawn fifteen horses                           2976
2930   {'action_sequence': [{ 'action_type': 'SPAWN',        2977
2931     'reference_object': {'has_name': [0, [2, 2]]},       2978
2932     'repeat': {'repeat_count': [0, [1, 1]],            2979
2933       'repeat_key': 'FOR'}}],                         2980
2934   'dialogue_type': 'HUMAN_GIVE_COMMAND' }

2935 dig a hole beneath the fence on the west side of the prison yard big enough for a 2983
2936   person to crawl through                      2984
2937   {'action_sequence': [{ 'action_type': 'DIG',          2985
2938     'location': {'location_type': 'REFERENCE_OBJECT', 2986
2939       'reference_object': {'has_name': [0, [5, 13]]},    2987
2940       'relative_direction': 'DOWN'},                  2988
2941     'repeat': {'stop_condition': {'condition_span': [0, 2989
2942       [14, 21]]}},                                2990
2943     'schematic': {'has_name': [0, [2, 2]]}}],           2991
2944   'dialogue_type': 'HUMAN_GIVE_COMMAND' }

2945 move trash outside                            2993
2946   {'action_sequence': [{ 'action_type': 'OTHERACTION', 2994
2947     'location': {'location_type': 'REFERENCE_OBJECT', 2995
2948       'reference_object': {},                      2996
2949       'relative_direction': 'OUTSIDE'},             2997
2950     'reference_object': {'has_name': [0, [1, 1]]}}],   2998
2951   'dialogue_type': 'HUMAN_GIVE_COMMAND' }

2952 add blocks to the right side of the house      2999
2953   {'action_sequence': [{ 'action_type': 'BUILD',      2999

```

```

3000         'location': {'location_type': 'REFERENCE_OBJECT',
3001                         'reference_object': {'has_name': [0,
3002                                         [8, 8]]},
3003                         'relative_direction': 'RIGHT'},
3004                         'schematic': {'has_name': [0, [1, 1]]}}}],
3005     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3006 dig a small hole
3007 {'action_sequence': [{('action_type': 'DIG',
3008                         'schematic': {'has_name': [0, [3, 3]],
3009                                         'has_size': [0, [2, 2]]}}}],
3010     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3011 sing
3012 {'action_sequence': [{('action_type': 'OTHERACTION')},
3013                         'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3014 hello bot move the trash cans from the backyard to the front yard
3015 {'action_sequence': [{('action_type': 'OTHERACTION',
3016                         'location': {'location_type': 'REFERENCE_OBJECT',
3017                                         'reference_object_1': {'has_name': [0,
3018                                         [6, 8]]},
3019                                         'reference_object_2': {'has_name': [0,
3020                                         [9,
3021                                         12]]}},
3022                                         'relative_direction': 'BETWEEN'},
3023                                         'reference_object': {'has_name': [0, [4, 5]]},
3024                                         'repeat': {'repeat_key': 'ALL'}},
3025     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3026 plant the tree
3027 {'action_sequence': [{('action_type': 'OTHERACTION',
3028                         'reference_object': {'has_name': [0, [2, 2]]}}}],
3029     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3030 dig in front of the red pyramid on the side facing me
3031 {'action_sequence': [{('action_type': 'DIG',
3032                         'location': {'location_type': 'REFERENCE_OBJECT',
3033                                         'reference_object': {'has_name': [0,
3034                                         [5, 11]]}},
3035                                         'relative_direction': 'FRONT'}},
3036     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3037 please complete what you were building
3038 {'action_sequence': [{('action_type': 'FREEBUILD',
3039                         'reference_object': {'has_name': [0, [2, 5]]}}}],
3040     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3041 repeat what i am going to say
3042 {'action_sequence': [{('action_type': 'OTHERACTION',
3043                         'reference_object': {'has_name': [0, [1, 6]]}}}],
3044     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3045 dig ten blocks down
3046 {'action_sequence': [{('action_type': 'DIG',
3047                         'location': {'location_type': 'AGENT_POS',
3048                                         'relative_direction': 'DOWN',
3049                                         'steps': [0, [1, 1]]})}],
3050     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

3051 copy the building next to the mine
3052 {'action_sequence': [{('action_type': 'BUILD',
3053                         'location': {'location_type': 'REFERENCE_OBJECT',
3054                                         'reference_object': {'has_name': [0,
3055                                         [6, 6]]},
3056                                         'relative_direction': 'NEAR'},
3057                                         'reference_object': {'has_name': [0, [2, 2]],
3058                                         'location': {'location_type': 'REFERENCE_OBJECT',
3059                                         'relative_direction': 'NEAR'}}}],
3060     'dialogue_type': 'HUMAN_GIVE_COMMAND' }

```

```

3100
3101 what is the weather tomorrow
3102 {'dialogue_type': 'GET_MEMORY'}
3103
3104 can you mine me minerals
3105 {'action_sequence': [{{'action_type': 'DIG',
3106 'repeat': {'stop_condition': {'block_type': [0, [4, 4]],
3107 'condition_type': 'ADJACENT_TO_BLOCK_TYPE
3108 }}}],,
3109 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3110
3111 Some examples from interactive data:
3112
3113 ttt
3114 {'dialogue_type': 'NOOP'}
3115
3116 build granite
3117 {'action_sequence': [{{'action_type': 'BUILD',
3118 'schematic': {'has_name': [0, [1, 1]]}}}],
3119 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3120
3121 build me a gate
3122 {'action_sequence': [{{'action_type': 'BUILD',
3123 'schematic': {'has_name': [0, [3, 3]]}}}],
3124 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3125
3126 build a wall to 69 64 30
3127 {'action_sequence': [{{'action_type': 'BUILD',
3128 'location': {'coordinates': [0, [4, 6]],
3129 'location_type': 'COORDINATES'},
3130 'schematic': {'has_name': [0, [2, 2]]}}}],
3131 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3132
3133 and add it to the tree
3134 {'action_sequence': [{{'action_type': 'OTHERACTION',
3135 'reference_object': {'has_name': [0, [5, 5]]}}}],
3136 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3137
3138 attack
3139 {'action_sequence': [{{'action_type': 'OTHERACTION'}},
3140 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3141
3142 what commands
3143 {'dialogue_type': 'GET_MEMORY'}
3144
3145 mine for gold
3146 {'action_sequence': [{{'action_type': 'DIG',
3147 'repeat': {'stop_condition': {'block_type': [0, [2, 2]],
3148 'condition_type': 'ADJACENT_TO_BLOCK_TYPE
3149 }}}],,
3150 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3151
3152 how do you make a furnace
3153 {'dialogue_type': 'GET_MEMORY'}
3154
3155 how do i name
3156 {'dialogue_type': 'GET_MEMORY'}
3157
3158 move here
3159 {'action_sequence': [{{'action_type': 'MOVE',
3160 'location': {'contains_coreference': 'yes'}}}],
3161 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3162
3163 fill
3164 {'action_sequence': [{{'action_type': 'FILL'}},
3165 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3166
3167 make the wall taller
3168 {'action_sequence': [{{'action_type': 'FREEBUILD',
3169 'reference_object': {'has_name': [0, [2, 2]]}}}],
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
```

```

3200     'dialogue_type': 'HUMAN_GIVE_COMMAND' } 3250
3201
3202     walk 3251
3203     {'action_sequence': [{}{'action_type': 'MOVE'}], 3252
3204     'dialogue_type': 'HUMAN_GIVE_COMMAND' } 3253
3205
3206     place a glass arch 3254
3207     {'action_sequence': [{}{'action_type': 'SPAWN', 3255
3208         'reference_object': {'has_block_type': [0, [2, 2]], 3256
3209             'has_name': [0, [3, 3]]}}], 3257
3210     'dialogue_type': 'HUMAN_GIVE_COMMAND' } 3258
3211
3212     build a pool 3259
3213     {'action_sequence': [{}{'action_type': 'BUILD', 3260
3214         'schematic': {'has_name': [0, [2, 2]]}}], 3261
3215     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3216
3217     collect the mushrooms here 3262
3218     {'action_sequence': [{}{'action_type': 'OTHERACTION', 3263
3219         'location': {'contains_coreference': 'yes'}, 3264
3220             'reference_object': {'has_name': [0, [2, 2]]}}], 3265
3221     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3222
3223     build wall 7 3266
3224     {'action_sequence': [{}{'action_type': 'BUILD', 3267
3225         'schematic': {'has_height': [0, [2, 2]], 3268
3226             'has_name': [0, [1, 1]]}}], 3269
3227     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3228
3229     build a bunker 3270
3230     {'action_sequence': [{}{'action_type': 'BUILD', 3271
3231         'schematic': {'has_name': [0, [2, 2]]}}], 3272
3232     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3233
3234     build a castle 3273
3235     {'action_sequence': [{}{'action_type': 'BUILD', 3274
3236         'schematic': {'has_name': [0, [2, 2]]}}], 3275
3237     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3238
3239     kill pig 3276
3240     {'action_sequence': [{}{'action_type': 'DESTROY', 3277
3241         'reference_object': {'has_name': [0, [1, 1]]}}], 3278
3242     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3243
3244     make a crafting table 3279
3245     {'action_sequence': [{}{'action_type': 'BUILD', 3280
3246         'schematic': {'has_name': [0, [2, 3]]}}], 3281
3247     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3248
3249     go to house 3282
3250     {'action_sequence': [{}{'action_type': 'MOVE', 3283
3251         'location': {'location_type': 'REFERENCE_OBJECT', 3284
3252             'reference_object': {'has_name': [0, 3285
3253                 [2, 2]]}}], 3286
3254     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3255
3256     dig a hole 5 blocks long and 5 blocks wide 3287
3257     {'action_sequence': [{}{'action_type': 'DIG', 3288
3258         'schematic': {'has_length': [0, [3, 3]], 3289
3259             'has_name': [0, [2, 2]], 3290
3260                 'has_width': [0, [7, 7]]}}], 3291
3261     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3262
3263     build a river 3292
3264     {'action_sequence': [{}{'action_type': 'BUILD', 3293
3265         'schematic': {'has_name': [0, [2, 2]]}}], 3294
3266     'dialogue_type': 'HUMAN_GIVE_COMMAND' }
3267
3268     build a 5 by 5 tower 3295

```

```

3300 {'action_sequence': [{}{'action_type': 'BUILD',
3301     'schematic': {'has_height': [0, [2, 2]],
3302         'has_name': [0, [5, 5]],
3303         'has_width': [0, [4, 4]]}}],
3304     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3305
3306 i am going now
3307 {'dialogue_type': 'NOOP'}
3308
3309 destroy
3310 {'action_sequence': [{}{'action_type': 'DESTROY'}],
3311     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3312
3313 chop wood
3314 {'action_sequence': [{}{'action_type': 'DESTROY',
3315     'reference_object': {'has_name': [0, [1, 1]]}}},
3316     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3317
3318 move to 59.7/67.9/-2.9
3319 {'action_sequence': [{}{'action_type': 'MOVE',
3320     'location': {'coordinates': [0, [2, 2]],
3321         'location_type': 'COORDINATES'}}],
3322     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3323
3324 build 10 x 12 wll
3325 {'action_sequence': [{}{'action_type': 'BUILD',
3326     'schematic': {'has_height': [0, [1, 1]],
3327         'has_name': [0, [4, 4]],
3328         'has_width': [0, [3, 3]]}}},
3329     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3330
3331 buy a pyramid
3332 {'action_sequence': [{}{'action_type': 'OTHERACTION',
3333     'reference_object': {'has_name': [0, [2, 2]]}}},
3334     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3335
3336 build city
3337 {'action_sequence': [{}{'action_type': 'BUILD',
3338     'schematic': {'has_name': [0, [1, 1]]}}},
3339     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3340
3341 finish walls
3342 {'action_sequence': [{}{'action_type': 'FREEBUILD',
3343     'reference_object': {'has_name': [0, [1, 1]]}}},
3344     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3345
3346 build windows there
3347 {'action_sequence': [{}{'action_type': 'BUILD',
3348     'location': {'contains_coreference': 'yes'},
3349     'schematic': {'has_name': [0, [1, 1]]}}},
3350     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3351
3352 build me somethign
3353 {'action_sequence': [{}{'action_type': 'BUILD',
3354     'schematic': {'has_name': [0, [2, 2]]}}},
3355     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3356
3357 what is it
3358 {'dialogue_type': 'GET_MEMORY'}
3359
3360 turn around
3361 {'action_sequence': [{}{'action_type': 'MOVE',
3362     'location': {'location_type': 'AGENT_POS',
3363         'relative_direction': 'AROUND'}}],
3364     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3365
3366 kill it
3367 {'action_sequence': [{}{'action_type': 'DESTROY',
3368     'reference_object': {'contains_coreference': 'yes'}}],
3369     'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
```

```

3400
3401     place cows
3402         {'action_sequence': [{}{'action_type': 'SPAWN',
3403             'reference_object': {'has_name': [0, [1, 1]]}}],,
3404             'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3405
3406     give me a diamond pickaxe
3407         {'action_sequence': [{}{'action_type': 'OTHERACTION',
3408             'reference_object': {'has_block_type': [0, [3, 3]],
3409                 'has_name': [0, [4, 4]]}}],
3410             'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3411
3412     build me cool
3413         {'action_sequence': [{}{'action_type': 'BUILD',
3414             'schematic': {'has_name': [0, [2, 2]]}}],
3415             'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3416
3417     build here solid 4 x 4 x 4 glass
3418         {'action_sequence': [{}{'action_type': 'BUILD',
3419             'location': {'contains_coreference': 'yes'},
3420                 'schematic': {'has_block_type': [0, [8, 8]],
3421                     'has_height': [0, [3, 3]],
3422                         'has_tag': [0, [2, 2]],
3423                             'has_thickness': [0, [7, 7]],
3424                                 'has_width': [0, [5, 5]]}},
3425             'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3426
3427     build a stained glass cube here
3428         {'action_sequence': [{}{'action_type': 'BUILD',
3429             'location': {'contains_coreference': 'yes'},
3430                 'schematic': {'has_block_type': [0, [2, 3]],
3431                     'has_name': [0, [4, 4]]}},
3432             'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3433
3434     build hollow rectanguloid
3435         {'action_sequence': [{}{'action_type': 'BUILD',
3436             'schematic': {'has_name': [0, [2, 2]],
3437                 'has_tag': [0, [1, 1]]}},
3438             'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3439
3440     copy that
3441         {'action_sequence': [{}{'action_type': 'BUILD',
3442             'reference_object': {'contains_coreference': 'yes'}},
3443                 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3444
3445     grow trees
3446         {'action_sequence': [{}{'action_type': 'BUILD',
3447             'schematic': {'has_name': [0, [1, 1]]}},
3448                 'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3449
3450     name this a mushroom
3451         {'dialogue_type': 'PUT_MEMORY',
3452             'filters': {'reference_object': {'contains_coreference': 'yes'}},
3453                 'upsert': {'memory_data': {'has_tag': [0, [3, 3]], 'memory_type': 'TRIPLE'}}}
3454
3455     build brick wall
3456         {'action_sequence': [{}{'action_type': 'BUILD',
3457             'schematic': {'has_block_type': [0, [1, 1]],
3458                 'has_name': [0, [2, 2]]}},
3459             'dialogue_type': 'HUMAN_GIVE_COMMAND'}
3460
3461     build here solid glass 4 x 4 x 4
3462         {'action_sequence': [{}{'action_type': 'BUILD',
3463             'location': {'contains_coreference': 'yes'},
3464                 'schematic': {'has_block_type': [0, [3, 3]],
3465                     'has_height': [0, [4, 4]],
3466                         'has_tag': [0, [2, 2]],
3467                             'has_thickness': [0, [8, 8]],
3468                                 'has_width': [0, [6, 6]]}},
3469             'dialogue_type': 'HUMAN_GIVE_COMMAND'}

```