Interactive Language Learning by Question Answering

Xingdi Yuan[♡*] **Marc-Alexandre Côté**[♡*] **Jie Fu**[♣♠] **Zhouhan Lin**[♢♠]
Christopher Pal[♣♠] **Yoshua Bengio**[♢♠] **Adam Trischler**[♡]
[♡]Microsoft Research, Montréal [♣]Polytechnique Montréal
[♢]Université de Montréal [♠] Mila
eric.yuan@microsoft.com macote@microsoft.com

Abstract

Humans observe and interact with the world to acquire knowledge. However, most existing machine reading comprehension (MRC) tasks miss the interactive, information-seeking component of comprehension. Such tasks present models with static documents that contain all necessary information, usually concentrated in a single short substring. Thus, models can achieve strong performance through simple word- and phrase-based pattern matching. We address this problem by formulating a novel text-based question answering task: Question Answering with Interactive Text (QAit)[1]. In QAit, an agent must interact with a partially observable text-based environment to gather information required to answer questions. QAit poses questions about the existence, location, and attributes of objects found in the environment. The data is built using a text-based game generator that defines the underlying dynamics of interaction with the environment. We propose and evaluate a set of baseline models for the QAit task that includes deep reinforcement learning agents. Experiments show that the task presents a major challenge for machine reading systems, while humans solve it with relative ease.

1 Introduction

The research community has defined the task of machine reading comprehension (MRC) to teach machines to read and understand text. In most MRC tasks, given a knowledge source (usually a text document) and a question on its content, a model is required to answer the question either by pointing to words in the source or by generating a text string. Recent years have seen a flourishing of MRC works, including the release of numerous datasets (Rajpurkar et al., 2016; Trischler et al., 2016; Nguyen et al., 2016; Reddy et al., 2018; Yang et al., 2018) and a host of neural models that address them (Wang and Jiang, 2016; Wang et al., 2017; Chen et al., 2017).

Game	You find yourself in a backyard. You make out a patio table. You see a patio chair. It's stylish. In a corner you can see a BBQ. There's an open screen door leading south.
Quest	Is ghargh edible?
Agent	> **Qapla'**
Game	That's not a verb I recognize.
Agent	> **eat ghargh**
Game	You can't see any such thing.
Agent	> **go south**
Game	You've just entered a kitchen. You can see an open fridge here. It is empty! You see a closed oven. Look over there, it's a counter. On the counter you can see a diced potato, a red ghargh and a knife. A an open door is leading north.
Agent	> **eat ghargh**
Game	You need to take the red ghargh first.
Agent	> **take red ghargh**
Game	You take the red ghargh from the counter.
Agent	> **eat ghargh**
Game	You eat the red ghargh. Not bad.
Agent	> **wait**
Agent	> **answer: yes**

Table 1: Example of a player interacting with an environment to gather knowledge and answer a question.

Most existing MRC datasets, as a consequence of their construction, encourage models to do fairly shallow phrase and word matching between the question and the knowledge source. There are three reasons for this. First, questions tend to paraphrase or permute the text that expresses the answer, making pattern matching an effective strategy. On the SQuAD development set, for example, the average precision, recall, and F_1 score between questions and the sentences containing answer spans are 0.409, 0.271, 0.301, respectively. Exploiting this overlap, Trischler et al.

[*] Equal contribution.

[1]The dataset and implementation of our baseline agents are publicly available at https://github.com/xingdi-eric-yuan/qait_public.

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 2796–2813, Hong Kong, China, November 3–7, 2019.

(2016) show that a simple Information Retrieval method can achieve high sentence-level accuracy on SQuAD.

Second, the information that supports predicting the answer from the source is often *fully observed*: the source is static, sufficient, and presented in its entirety. This does not match the information-seeking procedure that arises in answering many natural questions (Kwiatkowski et al., 2019), nor can it model the way humans observe and interact with the world to acquire knowledge.

Third, most existing MRC studies focus on *declarative* knowledge — the knowledge of facts or events that can be stated explicitly (i.e., declared) in short text snippets. Given a static description of an entity, declarative knowledge can often be extracted straightforwardly through pattern matching. For example, given the EMNLP website text, the conference deadline can be extracted by matching against a date mention. This focus overlooks another essential category of knowledge — *procedural* knowledge. Procedural knowledge entails executable sequences of actions. These might comprise the procedure for tying ones shoes, cooking a meal, or gathering new declarative knowledge. The latter will be our focus in this work. As an example, a more general way to determine EMNLP's deadline is to open a browser, head to the website, and then match against the deadline mention; this involves executing several mouse and keyboard interactions.

In order to teach MRC systems *procedures* for question answering, we propose a novel task: Question Answering with Interactive Text (QAit). Given a question $q \in Q$, rather than presenting a model with a static document $d \in D$ to read, QAit requires the model to interact with a partially observable environment $e \in E$ over a sequence of turns. The model must collect and aggregate evidence as it interacts, then produce an answer a to q based on its experience.

In our case, the environment e is a text-based game with no explicit objective. The game places an agent in a simple modern house populated by various everyday objects. The agent may explore and manipulate the environment by issuing text commands. An example is shown in Table 1. We build a corpus of related text-based games using a generator from Côté et al. (2018), which enables us to draw games from a controlled distribution. This means there are random variations across the environment set E, in map layouts and in the existence, location, and names of objects, etc. Consequently, an agent cannot answer questions merely by memorizing games it has seen before. Because environments are partially observable (i.e., not all necessary information is available at a single turn), an agent must take a sequence of decisions – analogous to following a search and reasoning procedure – to gather the required information. The learning target in QAit is thus not the declarative knowledge a itself, but the *procedure* for arriving at a by collecting evidence.

The main contributions of this work are as follows:

1. We introduce a novel MRC dataset, QAit, which focuses on *procedural* knowledge. In it, an agent interacts with an environment to discover the answer to a given question.

2. We introduce to the MRC domain the practice of generating training data on the fly. We sample training examples from a distribution; hence, an agent is highly unlikely to encounter the same training example more than once. This helps to prevent overfitting and rote memorization.

3. We evaluate a collection of baseline agents on QAit, including state-of-the-art deep reinforcement learning agents and humans, and discuss limitations of existing approaches.

2 The QAit Dataset

2.1 Overview

We make the question answering problem interactive by building text-based games along with relevant question-answer pairs. We use TextWorld (Côté et al., 2018) to generate these games. Each interactive environment is composed of multiple locations with paths connecting them in a randomly drawn graph. Several interactable objects are scattered across the locations. A player sends text commands to interact with the world, while the game's interpreter only recognizes a small subset of all possible command strings (we call these the valid commands). The environment changes state in response to a valid command and returns a string of text feedback describing the change.

The underlying game dynamics arise from a set of objects (e.g., doors) that possess attributes (e.g.,

	edible	drinkable	portable	openable	cuttable	sharp	heat_source	cookable	holder
Butter knife			1			1			
Oven				1			1		1
Raw chicken			1		1			1	
Fried chicken	1		1		1			1	

Table 2: Supported attributes along with examples.

doors are openable), and a set of rules (e.g., opening a closed door makes the connected room accessible). The supported attributes are shown in Table 2, while the rules can be inferred from the list of supported commands (see Appendix C). Note that player interactions might affect an object's attributes. For instance, cooking a piece of *raw chicken* on the stove with a frying pan makes it edible, transforming it into *fried chicken*.

In each game, the existence of objects, the location of objects, and their names are randomly sampled. Depending on the task, a name can be a made-up word. However, game dynamics are constant across all games – e.g., there will never be a drinkable heat source.

Text in QAit is generated by the TextWorld engine according to English templates, so it does not express the full variation of natural language. However, taking inspiration from the bAbI tasks (Weston et al., 2015), we posit that controlled simplifications of natural language are useful for isolating more complex reasoning behaviors.

2.2 Available Information

At every game step, the environment returns an *observation string* describing the information visible to the agent, as well as the *command feedback*, which is text describing the response to the previously issued command.

Optional Information: Since we have access to the underlying state representation of a generated game, various optional information can be made available. For instance, it is possible to access the subset of commands that are valid at the current game step. Other available meta-information includes all objects that exist in the game, plus their locations, attributes, and states.

During training, one is free to use any optional information to guide the agent's learning, e.g., to shape the rewards. However, at test time, only the *observation string* and the *command feedback* are available.

2.3 Question Types and Difficulty Levels

Using the game information described above, we can generate questions with known ground truth answers for any given game.

2.3.1 Question Types

For this initial version of QAit we consider three straightforward question types.

Location: ("Where is the can of soda?") Given an object name, the agent must answer with the name of the container that most directly holds the object. This can be either a location, a holder within a location, or the player's inventory. For example, if the can of soda is in a fridge which is in the kitchen, the answer would be "fridge".

Existence: ("Is there a raw egg in the world?") Given the name of an object, the agent must learn to answer whether the object exists in the game environment e.

Attribute: ("Is *ghargh* edible?") Given an object name and an attribute, the agent must answer with the value of the given attribute for the given object. Note that all attributes in our dataset are binary-valued. To discourage an agent from simply memorizing attribute values given an object name (Anand et al., 2018) (e.g., apples are always edible so agents can answer without interaction), we replace object names with unique, randomly drawn made-up words for this question type.

2.3.2 Difficulty Levels

To better analyze the limitations of learning algorithms and to facilitate curriculum learning approaches, we define two difficulty levels based on the environment layout.

Fixed Map: The map (location names and layout) is fixed across games. Random objects are distributed across the map in each game. Statistics for this game configuration are shown in Table 3.

Random Map: Both map layouts and objects are randomly sampled in each game.

2.4 Action Space

We describe the action space of QAit by splitting it into two subsets: information-gathering actions and question-answering actions.

	Fixed Map	Random Map
# Locations, N_r	6	$N_r \sim \text{Uniform}[2, 12]$
# Entities, N_e	$N_e \sim \text{Uniform}[3 \cdot N_r, 6 \cdot N_r]$	
Actions / Game	17	17
Modifiers / Game	18.5	17.7
Objects / Game	26.7	27.5
# Obs. Tokens	93.1	89.7

Table 3: Statistics of the QAit dataset. Numbers are averaged over 10,000 randomly sampled games.

Information Gathering The player generates text commands word by word to navigate through and interact with the environment. On encountering an object, the player must interact with it to discover its attributes. To succeed, an agent must map the feedback received from the environment, in text, to a useful state representation. This is a form of reading comprehension.

To make the QAit task more tractable, all text commands are triplets of the form {action, modifier, object} (e.g., `open wooden door`). When there is no ambiguity, the environment understands commands without modifiers (e.g., `eat apple` will result in eating the "red apple" provided it is the only apple in the player's inventory). We list all supported commands in Appendix C.

Each game provides a set of three lexicons that divide the full vocabulary into actions, modifiers, and objects. Statistics are shown in Table 3. A model can generate a command at each game step by, e.g., sampling from a probability distribution induced over each lexicon. This reduces the size of the action space compared to a sequential, free-form setting where a model can pick any vocabulary word at any generation step.

An agent decides when to stop interacting with the environment to answer the question by generating a special `wait` command [2]. However, the number of interaction steps is limited: we use 80 steps in all experiments. When an agent has exhausted its available steps, the game terminates and the agent is forced to answer the question.

Question Answering Currently, all QAit answers are one word. For existence and attribute questions, the answer is either `yes` or `no`; for location questions, the answer can be any word in an observation string.

[2]We call it "wait" because when playing multiple games in a batch, batched environment will terminate only when all agents have issued the terminating command. Before that, some agent will wait. This is analogous to paddings in natural language processing tasks.

2.5 Evaluation Settings and Metrics

We evaluate an agent's performance on QAit by its accuracy in answering questions. We propose three distinct settings for the evaluation.

Solving Training Games: We use QA accuracy during training, averaged over a window of training time, to evaluate an agent's training performance. We provide 5 training sets for this purpose with [1, 2, 10, 100, 500] games, respectively. Each game in these sets is associated with multiple questions.

Unlimited Games: We implement a setup where games are randomly generated on the fly during training, rather than selected from a finite set as above. The distribution we draw from is controlled by a few parameters: number of locations, number of objects, type of map, and a random seed. From the **fixed map** game distribution described in Table 3, more than 10^{40} different games can be drawn. This means that a game is unlikely to be seen more than once during training. We expect that only a model with strong generalization capabilities will perform well in this setting.

Zero-shot Evaluation: For each game setting and question type, we provide 500 held out games that are never seen during training, each with one question. These are used to benchmark generalization in models in a reproducible manner, no matter the training setting. This set is analogous to the test set used in traditional supervised learning tasks, and can be used in conjunction with any training setting.

3 Baseline Models

3.1 Random Baseline

Our simplest baseline does not interact with the environment to answer questions; it samples an answer word uniformly from the QA action space (`yes` and `no` for attribute and existence questions; all possible object names in the game for location questions).

3.2 Human Baseline

We conducted a study with 21 participants to explore how humans perform on QAit in terms of

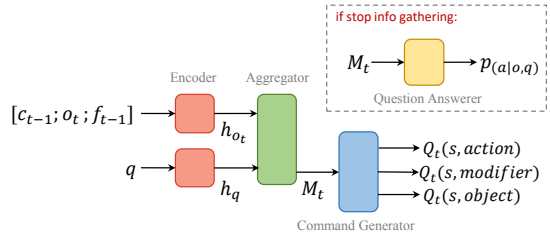

Figure 1: Overall architecture of our baseline agent.

QA accuracy. Participants played games they had not seen previously from a set generated by sampling 4 game-question pairs for each question type and difficulty level. The human results presented below always represent an average over 3 participants.

3.3 QA-DQN

We propose a neural baseline agent, QA-DQN, which takes inspiration from the work of Narasimhan et al. (2015) and Yu et al. (2018). The agent consists of three main components: an encoder, a command generator, and a question answerer. More precisely, at game step t, the encoder takes observation o_t and question q as input to generate hidden representations.[3] In the information gathering phase, the command generator generates Q-values for all action, modifier, and object words, with rankings of these Q-values used to generate text commands c_t. At any game step, the agent may decide to terminate information gathering and answer the question (or it is forced to do so if it has used up all of its moves). The question answerer uses the hidden representations at the final information-gathering step to generate a probability distribution over possible answers.

An overview of this architecture is shown in Figure 1 and full details are given in Appendix A.

3.3.1 Reward Shaping

We design the following two rewards to help QA-DQN learn more efficiently; both used for training the command generator. Note that these rewards are part of the design of QA-DQN, but are not used to evaluate its performance. Question answering accuracy is the only evaluation metric for QAit tasks.

Sufficient Information Bonus: To tackle QAit tasks, an intelligent agent should know when to stop interacting – it should stop as soon as it has gathered enough information to answer the question correctly. For guiding the agent to learn this behavior, we give an additional reward when the agent stops with sufficient information. Specifically, assuming the agent decides to stop at game step k:

- Location: reward is 1 if the entity mentioned in the question is a sub-string of o_k, otherwise it is 0. This means whenever an agent observes the entity, it has sufficient information to infer the entity's location.
- Existence: when the correct answer is `yes`, a reward of 1 is assigned only if the entity is a sub-string of o_k. When the correct answer is `no`, a reward between 0 and 1 is given. The reward value corresponds to the exploration coverage of the environment, i.e., how many locations the agent has visited, and how many containers have been opened.
- Attribute: we heuristically define a set of conditions to verify each attribute, and reward the agent based on its fulfilment of these conditions. For instance, determining if an object `x` is **sharp** corresponds to checking the outcome of a cut command (`slice`, `chop`, or `dice`) while holding the object `x` and a cuttable food item. If the outcome is successful then the object `x` is sharp otherwise it is not. Alternatively, if trying to take the object `x` results in a failure, then we can deduces it is not sharp as all sharp objects are portable. The list of conditions for each attribute used in our experiments is shown in Appendix D.

Episodic Discovery Bonus: Following Yuan et al. (2018), we use an episodic counting reward to encourage the agent to discover unseen game states. The agent is assigned a positive reward whenever it encounters a new state (in text-based games, states are simply represented as strings):

$$r(o_t) = \begin{cases} 1.0 & \text{if } n(o_t) = 1, \\ 0.0 & \text{otherwise,} \end{cases}$$

where $n(\cdot)$ is reset to zero after each episode.

3.3.2 Training Strategy

We apply different training strategies for the command generator and the question answerer.

Command Generation: Text-based games are sequential decision-making problems that can

[3] We concatenate o_t with the command generated at previous game step and the text feedback returned by the game, as described in Section 2.2.

be described naturally by partially observable Markov decision processes (POMDPs) (Kaelbling et al., 1998). We use the Q-Learning (Watkins and Dayan, 1992) paradigm to train our agent. Specifically, following Mnih et al. (2015), our Q-value function is approximated with a deep neural network. Beyond vanilla DQN, we also apply several extensions, such as Rainbow (Hessel et al., 2017), to our training process. Details are provided in Section 4.

Question Answering: During training, we push all question answering transitions (observation strings when interaction stops, question strings, ground-truth answers) into a replay buffer. After every 20 game steps, we randomly sample a mini-batch of such transitions from the replay buffer and train the question answerer with supervised learning (e.g., using negative log-likelihood (NLL) loss).

4 Experimental Results

In this section, we report experimental results by difficulty levels. All random baseline performance values are averaged over 100 different runs. In the following subsections, we use "DQN", "DDQN" and "Rainbow" to indicate QA-DQN trained with vanilla DQN, Double DQN with prioritized experience replay, and Rainbow, respectively. Training curves shown in the following figures represent a sliding-window average with a window size of 500. Moreover, each curve is the average of 3 random seeds. For evaluation, we selected the model with the random seed yielding the highest training accuracy to compute its accuracy on the test games. Due to space limitations, we only report some key results here. See Appendix E for the full experimental results.

4.1 Fixed Map

Figure 2 shows the training curves for the neural baseline agents when trained using 10 games, 500 games and the "unlimited" games settings. Table 4 reports their zero-shot test performance.

From Figure 2, we observe that when training data size is small (e.g., 10 games), our baseline agent trained with all the three RL methods successfully master the training games. Vanilla DQN and DDQN are particularly strong at memorizing the training games. When training on more games (e.g., 500 games and unlimited games), in which case memorization is more difficult, Rainbow agents start to show its superiority — it has similar accuracy as the other two methods, and even outperforms them in existence question type.

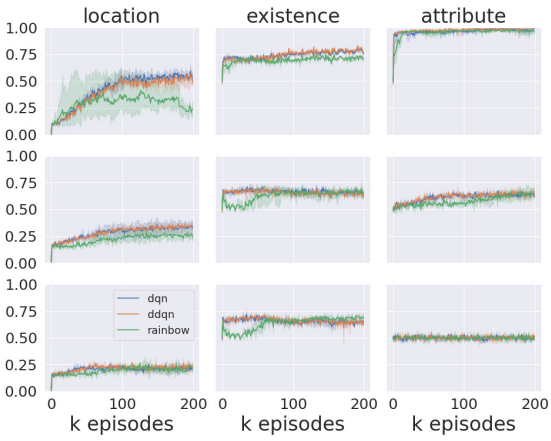

Figure 2: Training accuracy over episodes on **fixed map** setup. Upper row: 10 games; middle row: 500 games; lower row: unlimited games.

	Fixed Map			Random Map		
Model	Loc.	Exi.	Att.	Loc.	Exi.	Att.
Human	1.000	1.000	1.000	1.000	1.000	0.750
Random	0.027	0.497	0.496	0.034	0.500	0.499
10 games						
DQN	0.180	0.568	0.518	0.156	0.566	0.518
DDQN	0.188	0.566	0.516	0.142	0.606	0.500
Rainbow	0.156	0.590	0.520	0.144	0.586	**0.530**
500 games						
DQN	0.224	0.674	**0.534**	0.204	0.678	**0.530**
DDQN	0.218	0.626	0.508	0.222	0.656	0.486
Rainbow	0.190	0.656	0.496	0.172	0.678	0.494
unlimited games						
DQN	0.216	0.662	0.514	0.188	0.668	0.506
DDQN	0.258	0.628	0.480	0.206	**0.694**	0.482
Rainbow	**0.280**	**0.692**	0.514	**0.258**	0.686	0.470

Table 4: Agent performance on zero-shot test games when trained on 10 games, 500 games and "unlimited" games settings. Note Att. and Exi. are binary questions with expected accuracy of 0.5.

From Table 4 we see similar observation, when trained on 10 games and 500 games, DQN and DDQN performs better on test games but on the unlimited games setting, rainbow agent performs as good as them, and sometimes even better. We can also observe that our agents fail to generalize on attribute questions. In unlimited games setting as shown in Figure 2, all three agents produce an accuracy of 0.5; in zero-shot test as shown in Table 4, no agent performs significantly better than random. This suggests the agents memorize game-question-answer triples when data size is small, and fail to do so in unlimited games setting. This

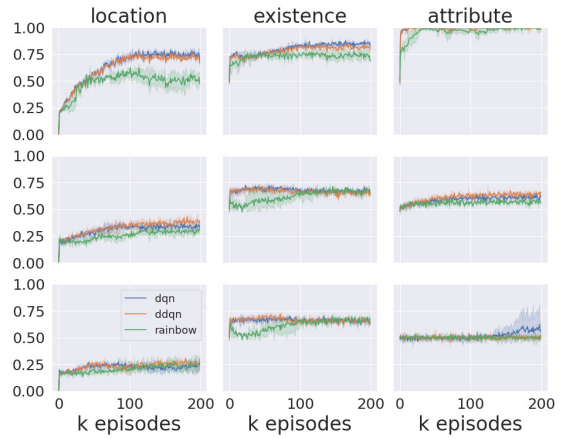

Figure 3: Training accuracy on the **random map** setup. Upper row: 10 games; middle row: 500 games; lower row: unlimited games.

can also be observed in Appendix E, where in attribute question experiments, the training accuracy is high, and sufficient information bonus is low (even close to 0).

4.2 Random Map

Figure 3 shows the training curves for the neural baseline agents when trained using 10 games, 500 games and "unlimited" games settings. The trends of our agents' performance on random map games are consistent with on fixed map games. However, because there exist easier games (as listed in Table 3, number of rooms is sampled between 2 and 12), agents show better training performance in such setting than **fixed map** setting in general.

Interestingly, we observe one of the DQN agent starts to learn in the unlimited games, attribute question setting. This may be because in games with smaller map size and less objects, there is a higher chance to accomplish some sub-tasks (e.g., it is easier to find an object when there are less rooms), and the agent learn such skills and apply them to similar tasks. Unfortunately, as shown in Table 4 that agent does not perform significantly better than random on test set. We expect with more training episodes, the agent can have a better generalization performance.

4.3 Question Answering Given Sufficient Information

The challenge in QAit is learning the interactive procedure for arriving at a state with the information needed to answer the question. We conduct the following experiments on **location** questions to investigate this challenge.

Model	Fixed Map	Random Map
Random	14.7	16.5
10 games		
DQN	95.7	97.5
DDQN	90.4	92.2
Rainbow	91.8	84.7
500 games		
DQN	91.8	94.4
DDQN	95.6	90.2
Rainbow	96.9	96.6
unlimited games		
DQN	100.0	100.0
DDQN	100.0	100.0
Rainbow	100.0	100.0

Table 5: Test performance given sufficient information.

Based on the results in Table 4, we compute an agent's test accuracy *only if* it has obtained sufficient information – i.e., when the sufficient information bonus is 1. Results shown in Table 5 support our assumption that the QA module can learn (and generalize) effectively to answer given sufficient information. Similarly, experiments show that when objects being asked about are in the current observation, the random baseline's performance goes up significantly as well. We report our baseline agents' question answering accuracy and sufficient information bonuses on all experiment settings in Appendix E.

4.4 Full Information Setup

To reframe the QAit games as a standard MRC task, we also designed an experimental setting that eliminates the need to gather information interactively. From a heuristic trajectory through the game environment that is guaranteed to observe sufficient information for q, we concatenate all observations into a static "document" d to build a $\{d, q, a\}$ triplet. A model then uses this fully observed document as input to answer the question. We split this data into training, validation, and test sets and follow the evaluation protocol for standard supervised MRC tasks. We take an off-the-shelf MRC model, Match-LSTM (Wang and Jiang, 2016), trained with negative log-likelihood loss as a baseline.

Unsurprisingly, Match-LSTM does fairly well on all 3 question types (86.4, 89.9 and 93.2 test accuracy on location, existence, and attribute questions, respectively). This implies that without the need to interact with the environment for information gathering, the task is simple enough that a word-matching model can answer questions with

high accuracy.

5 Related Work

5.1 MRC Datasets

Many large-scale machine reading comprehension and question answering datasets have been proposed recently. The datasets of Rajpurkar et al. (2016); Trischler et al. (2016) contain crowd-sourced questions based on single documents from Wikipedia and CNN news, respectively. Nguyen et al. (2016); Joshi et al. (2017); Dunn et al. (2017); Clark et al. (2018); Kwiatkowski et al. (2019) present question-answering corpora harvested from information retrieval systems, often containing multiple supporting documents for each question. This means a model must sift through a larger quantity of information and possibly reconcile competing viewpoints. Berant et al. (2013); Welbl et al. (2017); Talmor and Berant (2018) propose to leverage knowledge bases to generate question-answer pairs. Yang et al. (2018) focuses on questions that require multi-hop reasoning to answer, by building questions compositionally. Reddy et al. (2018); Choi et al. (2018) explore conversational question answering, in which a full understanding of the question depends on the conversation's history.

Most of these datasets focus on declarative knowledge and are static, with all information fully observable to a model. We contend that this setup, unlike QAit, encourages word matching. Supporting this contention, several studies highlight empirically that existing MRC tasks require little comprehension or reasoning. In Rychalska et al. (2018), it was shown that a question's main verb exerts almost no influence on the answer prediction: in over 90% of examined cases, swapping verbs for their antonyms does not change a system's decision. Jia and Liang (2017) show the accuracy of neural models drops from an average of 75% F_1 score to 36% F_1 when they manually insert adversarial sentences into SQuAD.

5.2 Interactive Environments

Several embodied or visual question answering datasets have been presented recently to address some of the problems of interest in our work, such as those of Brodeur et al. (2017); Das et al. (2017); Gordon et al. (2017). In contrast with these, our purely text-based environment circumvents challenges inherent to modelling interactions between separate data modalities. Furthermore, most visual question answering environments only support navigating and moving the camera as interactions. In text-based environments, however, it is relatively cheap to build worlds with complex interactions. This is because text enables us to model interactions abstractly without the need for, e.g., a costly physics engine.

Closely related to QAit is BabyAI (Chevalier-Boisvert et al., 2018). BabyAI is a gridworld environment that also features constrained language for generating simple home-based scenarios (i.e., instructions). However, observations and actions in BabyAI are not text-based. World of Bits (Shi et al., 2017) is a platform for training agents to interact with the internet to accomplish tasks like flight booking. Agents generally do not need to gather information in World of Bits, and the focus is on accomplishing tasks rather than answering questions.

5.3 Information Seeking

Information seeking behavior is an important capacity of intelligent systems that has been discussed for many years. Kuhlthau (2004) propose a holistic view of information search as a six-stage process. Schmidhuber (2010) discusses the connection between information seeking and formal notions of fun, creativity, and intrinsic motivation. Das et al. (2018) propose a model that continuously determines all entities' locations during reading and dynamically updates the associated representations in a knowledge graph. Bachman et al. (2016) propose a collection of tasks and neural methods for learning to gathering information efficiently in an environment.

To our knowledge, we are the first to consider interactive information-seeking tasks for question answering in worlds with complex dynamics. The QAit task was designed such that simple word matching methods do not apply, while more human-like information seeking models are encouraged.

6 Discussion and Future Work

Monitoring Information Seeking: In QAit, the only evaluation metric is question answering accuracy. However, the sufficient information bonus described in Section 3.3.1 is helpful for monitoring agents' ability to gather relevant information. We report its value for all experiments in

Appendix E. We observe that while the baseline agents can reach a training accuracy of 100% for answering attribute questions when trained on a few games, the sufficient information bonus is close to 0. This is a clear indication that the agent overfits to the question-answer mapping of the games rather than learning how to gather useful information. This aligns with our observation that the agent does not perform better than random on the unlimited games setting, because it fails to gather the needed information.

Challenges in QAit: QAit focuses on learning procedural knowledge from interactive environments, so it is natural to use deep RL methods to tackle it. Experiments suggest the dataset presents a major challenge for existing systems, including Rainbow, which set the state of the art on Atari games. As a simplified and controllable text-based environment, QAit can drive research in both the RL and language communities, especially where they intersect. Until recently, the RL community focused mainly on solving single environments (i.e., training and testing on the same game). Now, we see a shift towards solving multiple games and testing for generalization (Cobbe et al., 2018; Justesen et al., 2018). We believe QAit serves this purpose.

Templated Language: As QAit is based on TextWorld, it has the obvious limitation of using templated English. However, TextWorld provides approximately 500 *human-written* templates for describing rooms and objects, so some textual diversity exists, and since game narratives are generated compositionally, this diversity increases along with the complexity of a game. We believe simplified and controlled text environments offer a bridge to full natural language, on which we can isolate the learning of useful behaviors like information seeking and command generation. Nevertheless, it would be interesting to further diversify the language in QAit, for instance by having human writers paraphrase questions.

Future Work: Based on our present efforts to tackle QAit, we propose the following directions for future work.

A **structured memory** (e.g., a dynamic knowledge graph as proposed in Das et al. (2018); Ammanabrolu and Riedl (2019a)) could be helpful for explicitly memorizing the places and objects that an agent has observed. This is especially useful when an agent must revisit a location or object or should avoid doing so.

Likewise, a variety of **external knowledge** could be leveraged by agents. For instance, incorporating a pretrained language model could improve command generation by imparting knowledge of word and object affordances. In recent work, Hausknecht et al. (2019) show that pretrained modules together with handcrafted subpolicies help in solving text-based games, while Yin and May (2019) use BERT (Devlin et al., 2018) to inject 'weak common sense' into agents for text-based games. Ammanabrolu and Riedl (2019b) show that knowledge graphs and their associated neural encodings can be used as a medium for domain transfer across text-based games.

In finite game settings we observed significant overfitting, especially for attribute questions – as shown in Appendix E, our agent achieves high QA accuracy but low sufficient information bonus on the single-game setting. Sometimes attributes require long procedures to verify, and thus, we believe that denser rewards would help with this problem. One possible solution is to provide **intermediate rewards** whenever the agent achieves a sub-task.

Acknowledgments

The authors thank Romain Laroche, Rémi Tachet des Combes, Matthew Hausknecht, Philip Bachman, and Layla El Asri for insightful ideas and discussions. We thank Tavian Barnes, Wendy Tay, and Emery Fine for their work on the TextWorld framework. We also thank the anonymous EMNLP reviewers for their helpful feedback and suggestions.

References

Ammanabrolu, P. and Riedl, M. (2019a). Playing text-adventure games with graph-based deep reinforcement learning. In *NAACL*, pages 3557–3565, Minneapolis, Minnesota. Association for Computational Linguistics.

Ammanabrolu, P. and Riedl, M. (2019b). Transfer in deep reinforcement learning using knowledge graphs. *CoRR*, abs/1908.06556.

Anand, A., Belilovsky, E., Kastner, K., Larochelle, H., and Courville, A. C. (2018). Blindfold baselines for embodied QA. *CoRR*, abs/1811.05013.

Bachman, P., Sordoni, A., and Trischler, A. (2016). Towards information-seeking agents. *CoRR*, abs/1612.02605.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544. ACL.

Brodeur, S., Perez, E., Anand, A., Golemo, F., Celotti, L., Strub, F., Rouat, J., Larochelle, H., and Courville, A. C. (2017). Home: a household multimodal environment. *CoRR*, abs/1711.11017.

Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading wikipedia to answer open-domain questions. *CoRR*, abs/1704.00051.

Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., and Bengio, Y. (2018). Babyai: First steps towards grounded language learning with a human in the loop. *CoRR*, abs/1810.08272.

Choi, E., He, H., Iyyer, M., Yatskar, M., Yih, W., Choi, Y., Liang, P., and Zettlemoyer, L. (2018). Quac : Question answering in context. *CoRR*, abs/1808.07036.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think you have solved question answering? try arc, the AI2 reasoning challenge. *CoRR*, abs/1803.05457.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. (2018). Quantifying generalization in reinforcement learning. *CoRR*, abs/1812.02341.

Côté, M.-A., Kádár, A., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., Asri, L. E., Adada, M., Tay, W., and Trischler, A. (2018). Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532.

Das, A., Datta, S., Gkioxari, G., Lee, S., Parikh, D., and Batra, D. (2017). Embodied question answering. *CoRR*, abs/1711.11543.

Das, R., Munkhdalai, T., Yuan, X., Trischler, A., and McCallum, A. (2018). Building dynamic knowledge graphs from text using machine reading comprehension. *CoRR*, abs/1810.05682.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Dunn, M., Sagun, L., Higgins, M., Güney, V. U., Cirik, V., and Cho, K. (2017). Searchqa: A new q&a dataset augmented with context from a search engine. *CoRR*, abs/1704.05179.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2017). Noisy networks for exploration. *CoRR*, abs/1706.10295.

Gordon, D., Kembhavi, A., Rastegari, M., Redmon, J., Fox, D., and Farhadi, A. (2017). IQA: visual question answering in interactive environments. *CoRR*, abs/1712.03316.

Hausknecht, M., Loynd, R., Yang, G., Swaminathan, A., and Williams, J. D. (2019). Nail: A general interactive fiction agent.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298.

Jia, R. and Liang, P. (2017). Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. (2017). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *CoRR*, abs/1705.03551.

Justesen, N., Torrado, R. R., Bontrager, P., Khalifa, A., Togelius, J., and Risi, S. (2018). Procedural level generation improves generality of deep reinforcement learning. *CoRR*, abs/1806.10729.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kuhlthau, C. (2004). *Seeking Meaning: A Process Approach to Library and Information Services*. Information management, policy, and services. Libraries Unlimited.

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Kelcey, M., Devlin, J., Lee, K., Toutanova, K. N., Jones, L., Chang, M.-W., Dai, A., Uszkoreit, J., Le, Q., and Petrov, S. (2019). Natural questions: a benchmark for question answering research. *Transactions of the Association of Computational Linguistics*.

Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. (2018). Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Narasimhan, K., Kulkarni, T., and Barzilay, R. (2015). Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*.

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., and Deng, L. (2016). MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. In *NIPS-W*.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250.

Reddy, S., Chen, D., and Manning, C. D. (2018). Coqa: A conversational question answering challenge. *CoRR*, abs/1808.07042.

Rychalska, B., Basaj, D., Biecek, P., and Wroblewska, A. (2018). Does it care what you asked? understanding importance of verbs in deep learning qa system.

Schmidhuber, J. (2010). Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Trans. on Auton. Ment. Dev.*, 2(3):230–247.

Shi, T., Karpathy, A., Fan, L., Hernandez, J., and Liang, P. (2017). World of bits: An open-domain platform for web-based agents. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3135–3144, International Convention Centre, Sydney, Australia. PMLR.

Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *CoRR*, abs/1505.00387.

Talmor, A. and Berant, J. (2018). The web as a knowledge-base for answering complex questions. *CoRR*, abs/1803.06643.

Trischler, A., Wang, T., Yuan, X., Harris, J., Sordoni, A., Bachman, P., and Suleman, K. (2016). Newsqa: A machine comprehension dataset. *CoRR*, abs/1611.09830.

Wang, S. and Jiang, J. (2016). Machine comprehension using match-lstm and answer pointer. *CoRR*, abs/1608.07905.

Wang, W., Yang, N., Wei, F., Chang, B., and Zhou, M. (2017). Gated self-matching networks for reading comprehension and question answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 189–198. Association for Computational Linguistics.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

Welbl, J., Stenetorp, P., and Riedel, S. (2017). Constructing datasets for multi-hop reading comprehension across documents. *CoRR*, abs/1710.06481.

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merrinboer, B., Joulin, A., and Mikolov, T. (2015). Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. (2018). Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *CoRR*, abs/1809.09600.

Yin, X. and May, J. (2019). Learn how to cook a new recipe in a new house: Using map familiarization, curriculum learning, and common sense to learn families of text-based adventure games. *CoRR*, abs/1908.04777.

Yu, A. W., Dohan, D., Luong, M.-T., Zhao, R., Chen, K., Norouzi, M., and Le, Q. V. (2018). Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*.

Yuan, X., Côté, M., Sordoni, A., Laroche, R., des Combes, R. T., Hausknecht, M. J., and Trischler, A. (2018). Counting to explore and generalize in text-based games. *CoRR*, abs/1806.11525.

A Details of QA-DQN

Notations

In this section, we use *game step* t to denote one round of interaction between an agent with the QAit environment. We use o_t to denote text observation at game step t, and q to denote question text. We use L to refer to a linear transformation. Brackets $[\cdot;\cdot]$ denote vector concatenation.

A.1 Encoder

We use a transformer-based text encoder, which consists of an embedding layer, two stacks of transformer blocks (denoted as encoder transformer blocks and aggregation transformer blocks), and an attention layer.

In the embedding layer, we aggregate both word- and character-level information to produce a vector for each token in text. Specifically, word embeddings are initialized by the 300-dimensional fastText (Mikolov et al., 2018) word vectors trained on Common Crawl (600B tokens), they are fixed during training. Character level embedding vectors are initialized with 32-dimensional random vectors. A convolutional layer with 64 kernels of size 5 is then used to aggregate the sequence of characters. We use a max pooling layer on the character dimension, then a multi-layer perceptron (MLP) of output size 64 is used to aggregate the concatenation of word- and character-level representations. Highway network (Srivastava et al., 2015) is applied on top of this MLP. The resulting vectors are used as input to the encoding transformer blocks.

Each encoding transformer block consists of a stack of convolutional layers, a self-attention layer, and an MLP. In which, each convolutional layer has 64 filters, each kernel's size is 7, there are 2 such convolutional layers that share weights. In the self-attention layer, we use a block hidden size of 64, as well as a single head attention mechanism. Layernorm and dropout are applied after each component inside the block. We add positional encoding into each block's input. We use one layer of such an encoding block.

At a game step t, the encoder processes text observation o_t and question q, context aware encoding $h_{o_t} \in \mathbb{R}^{L^{o_t} \times H_1}$ and $h_q \in \mathbb{R}^{L^q \times H_1}$ are generated, where L^{o_t} and L^q denote number of tokens in o_t and q respectively, H_1 is 64. Following (Yu et al., 2018), we use an context-query attention layer to aggregate the two representations h_{o_t} and h_q.

Specifically, the attention layer first uses two MLPs to convert both h_{o_t} and h_q into the same space, the resulting tensors are denoted as $h'_{o_t} \in \mathbb{R}^{L^{o_t} \times H_2}$ and $h'_q \in \mathbb{R}^{L^q \times H_2}$, in which H_2 is 64.

Then, a tri-linear similarity function is used to compute the similarities between each pair of h'_{o_t} and h'_q items:

$$S = W[h'_{o_t}; h'_q; h'_{o_t} \odot h'_q], \tag{1}$$

where $\odot$ indicates element-wise multiplication, W is trainable parameters of size 64.

Softmax of the resulting similarity matrix S along both dimensions are computed, this produces S^A and S^B. Information in the two representations are then aggregated by:

$$\begin{aligned} h_{oq} &= [h'_{o_t}; P; h'_{o_t} \odot P; h'_{o_t} \odot Q], \\ P &= S_q h'^{\top}_q, \\ Q &= S_q S^{\top}_{o_t} h'^{\top}_{o_t}, \end{aligned} \tag{2}$$

where h_{oq} is aggregated observation representation.

On top of the attention layer, a stack of aggregation transformer blocks is used to further map the observation representations to action representations and answer representations. The structure of aggregation transformer blocks are the same as the encoder transformer blocks, except the kernel size of convolutional layer is 5, and the number of blocks is 3.

Let $M_t \in \mathbb{R}^{L^{o_t} \times H_3}$ denote the output of the stack of aggregation transformer blocks, where H_3 is 64.

A.2 Command Generator

The command generator takes the hidden representations M_t as input, it estimates Q-values for all action, modifier, and object words, respectively. It consists of a shared Multi-layer Perceptron (MLP) and three MLPs for each of the components:

$$\begin{aligned} R_t &= \text{ReLU}(L_{\text{shared}}(\text{mean}(M_t)), \\ Q_{t,action} &= L_{\text{action}}(R_t), \\ Q_{t,modifier} &= L_{\text{modifier}}(R_t), \\ Q_{t,object} &= L_{\text{object}}(R_t). \end{aligned} \tag{3}$$

In which, the output size of L_{shared} is 64; the dimensionalities of the other 3 MLPs are depending on the number of the amount of action, modifier

and object words available, respectively. The overall Q-value is the sum of the three components:

$$Q_t = Q_{t,action} + Q_{t,modifier} + Q_{t,object}. \quad (4)$$

A.3 Question Answerer

Similar to (Yu et al., 2018), we append an extra stacks of aggregation transformer blocks on top of the aggregation transformer blocks to compute answer positions:

$$\begin{aligned} U &= \text{ReLU}(L_0[M_t; M'_t]). \\ \beta &= \text{softmax}(L_1(U)). \end{aligned} \quad (5)$$

In which $M'_t \in \mathbb{R}^{L^{o_t} \times H_3}$ is output of the extra transformer stack, L_0, L_1 are trainable parameters with output size 64 and 1, respectively.

For location questions, the agent outputs β as the probability distribution of each word in observation o_t being the answer of the question.

For binary classification questions, we apply an MLP, which takes weighted sum of matching representations as input, to compute a probability distribution $p(y)$ over both possible answers:

$$\begin{aligned} D &= \sum_i (\beta^i \cdot M'_t), \\ p(y) &= \text{softmax}(L_4(\tanh(L_3(D)))). \end{aligned} \quad (6)$$

Output size of L_3 and L_4 are 64 and 2, respectively.

A.4 Deep Q-Learning

In a text-based game, an agent takes an action a[4] in state s by consulting a state-action value function $Q(s, a)$, this value function is as a measure of the action's expected long-term reward. Q-Learning helps the agent to learn an optimal $Q(s, a)$ value function. The agent starts from a random Q-function, it gradually updates its Q-values by interacting with environment, and obtaining rewards. Following Mnih et al. (2015), the Q-value function is approximated with a deep neural network.

We make use of a replay buffer. During playing the game, we cache all transitions into the replay buffer without updating the parameters. We periodically sample a random batch of transitions from the replay buffer. In each transition, we update the parameters θ to reduce the discrepancy between the predicted value of current state $Q(s_t, a_t)$ and the expected Q-value given the reward r_t and the value of next state $\max_a Q(s_{t+1}, a)$.

We minimize the temporal difference (TD) error, δ:

$$\delta = Q(s_t, a_t) - (r_t + \gamma \max_a Q(s_{t+1}, a)), \quad (7)$$

in which, γ indicates the discount factor. Following the common practice, we use the Huber loss to minimize the TD error. For a randomly sampled batch with batch size B, we minimize:

$$\begin{aligned} \mathcal{L} &= \frac{1}{|B|} \sum \mathcal{L}(\delta), \\ \text{where } \mathcal{L}(\delta) &= \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases} \end{aligned} \quad (8)$$

As described in Section 3.3.1, we design the sufficient information bonus to teach an agent to stop as soon as it has gathered enough information to answer the question. Therefore we assign this reward at the game step where the agent generates **`wait`** command (or it is forced to stop).

It is worth mentioning that for attribute type questions (considerably the most difficult question type in QAit, where the training signal is very sparse), we provide extra rewards to help QA-DQN to learn.

Specifically, we take a reward similar to as used in location questions: 1.0 if the agent has observed the object mentioned in the question. we also use a reward similar to as used in existence questions: the agent is rewarded by the coverage of its exploration. The two extra rewards are finally added onto the sufficient information bonus for attribute question, both with coefficient of 0.1.

B Implementation Details

During training with vanilla DQN, we use a replay memory of size 500,000. We use ϵ-greedy, where the value of ϵ anneals from 1.0 to 0.1 within 100,000 episodes. We start updating parameters after 1,000 episodes of playing. We update our network after every 20 game steps. During updating, we use a mini-batch of size 64. We use *Adam* (Kingma and Ba, 2014) as the step rule for optimization, The learning rate is set to 0.00025.

When our agent is trained with Rainbow algorithm, we follow Hessel et al. (2017) on most of the hyper-parameter settings. The four MLPs L_{shared}, L_{action}, L_{modifier} and L_{object} as described

[4] In our case, a is a triplet contains {action, modifier, object} as described in Section 2.4.

in Eqn. 3 are Noisy Nets layers (Fortunato et al., 2017) when the agent is trained in Rainbow setting. Detailed hyper-parameter setting of our Rainbow agent are shown in Table 6.

Parameter	Value
Exploration ϵ	0
Noisy Nets σ_0	0.5
Target Network Period	1000 episodes
Multi-step returns n	$n \sim \text{Uniform}[1, 3]$
Distributional atoms	51
Distributional min/max values	[-10, 10]

Table 6: Hyper-parameter setup for rainbow agent.

The model is implemented using PyTorch (Paszke et al., 2017).

C Supported Text Commands

All supported text commands are listed in Table 7.

D Heuristic Conditions for Attribute Questions

Here, we derived some heuristic conditions to determine when an agent has gathered enough information to answer a given attribute question. Those conditions are used as part of the reward shaping for our proposed agent (Section 3.3.1). In Table 8, for each attribute we list all the commands for which their outcome (pass or fail) gives enough information to answer the question correctly. Also, in order for a command's outcome to be informative, each command needs to be executed while some state conditions hold. For example, to determine if an object is indeed a **heat_source**, the agent needs to try to cook something that is cookable and uncooked while standing next to the given object.

E Full results

We provide full results of our agents on **fixed map** games in Table 9, and provide full results of our agents on **random map** games in Table 10. To help investigating the generalizability of the sufficient information bonus we used in our proposed agent, we also report the rewards during both training and test phases. Note during test phase, we do not update parameters with the rewards.

Command	Description
look	describe the current location
inventory	display the player's inventory
go ⟨**dir**⟩	move the player to north, east, south, or west
examine ...	examine something more closely
open ...	open a door or a container
close ...	close a door or a container
eat ...	eat edible object
drink ...	drink drinkable object
drop ...	drop an object on the floor
take ...	take an object from the floor, a container, or a supporter
put ...	put an object onto a supporter (supporter must be present at the location)
insert ...	insert an object into a container (container must be present at the location)
cook ...	cook an object (heat source must be present at the location)
slice ...	slice cuttable object (a sharp object must be in the player's inventory)
chop ...	chop cuttable object (a sharp object must be in the player's inventory
dice ...	dice cuttable object (a sharp object must be in the player's inventory)
wait	stop interaction

Table 7: Supported command list.

Attribute	Command	State	Pass	Fail	Explanation
sharp	**cut** *cuttable*	holding (*cuttable*) & uncut (*cuttable*) & holding (object)	1	1	Trying to cut something cuttable that hasn't been cut yet while holding the object.
	take object	reachable(object)	0	1	Sharp objects should be portable.
cuttable	**cut** object	holding (object) & holding (*sharp*)	1	1	Trying to cut the object while holding something sharp.
	take object	reachable (object)	0	1	Cuttable object should be portable.
edible	**eat** object	holding (object)	1	1	Trying to eat the object.
	take object	reachable (object)	0	1	Edible objects should be portable.
drinkable	**drink** object	holding (object)	1	1	Trying to drink the object.
	take object	reachable (object)	0	1	Drinkable objects should be portable.
holder	**–**	on (*portable*, object)	1	0	Observing object(s) on a supporter.
		in (*portable*, object)	1	0	Observing object(s) inside a container.
	take object	reachable (object)	1	0	Holder objects should not be portable.
portable	**–**	holding (object)	1	0	Holding the object means it is portable.
	take object	reachable (object)	1	1	Portable objects can be taken.
heat_source	**cook** *cookable*	holding (*cookable*) & uncooked (*cookable*) & reachable (object)	1	1	Trying to cook something cookable that hasn't been cooked yet while being next to the object.
	take object	reachable (object)	1	0	Heat source objects should not be portable.
cookable	**cook** object	holding (object) & reachable (*heat_source*)	1	1	Trying to cook the object while being next to a heat source.
	take object	reachable(object)	0	1	Cookable objects should be portable.
openable	**open** object	reachable (object) & closed (object)	1	1	Trying to open the closed object.
	close object	reachable (object) & open (object)	1	1	Trying to close the open object.

Table 8: Heuristic conditions for determining whether the agent has enough information to answer a given attribute question. We use "object" to refer to the object mentioned in the question. Words in italics represents placeholder that can be replaced by any object from the environment that has the appropriate attribute (e.g. carrot could be used as a *cuttable*). The columns Pass and Fail represent how much reward the agent will receive given the corresponding command's outcome (resp. success or failure). NB: **cut** can mean any of the following commands: **slice**, **dice**, or **chop**

Model	Location		Existence		Attribute	
	Train	Test	Train	Test	Train	Test
Human	–	1.000	–	1.000	–	1.000
Random	–	0.027	–	0.497	–	0.496
1 game						
DQN	0.972(0.972)	0.122(0.160)	1.000(0.881)	0.628(0.124)	1.000(0.049)	0.500(0.035)
DDQN	0.960(0.960)	0.156(0.178)	1.000(0.647)	0.624(0.148)	1.000(0.023)	0.498(0.033)
Rainbow	0.562(0.562)	0.164(0.178)	1.000(0.187)	0.616(0.083)	1.000(0.049)	0.516(0.039)
2 games						
DQN	0.698(0.698)	0.168(0.182)	0.948(0.700)	0.574(0.136)	1.000(0.011)	0.510(0.028)
DDQN	0.702(0.702)	0.172(0.178)	0.882(0.571)	0.550(0.109)	1.000(0.098)	0.508(0.036)
Rainbow	0.734(0.734)	0.160(0.168)	0.878(0.287)	0.616(0.085)	1.000(0.030)	0.524(0.022)
10 games						
DQN	0.654(0.654)	0.180(0.188)	0.822(0.390)	0.568(0.156)	1.000(0.055)	0.518(0.030)
DDQN	0.608(0.608)	0.188(0.208)	0.842(0.479)	0.566(0.128)	1.000(0.064)	0.516(0.036)
Rainbow	0.616(0.616)	0.156(0.170)	0.768(0.266)	0.590(0.131)	0.998(0.059)	0.520(0.023)
100 games						
DQN	0.498(0.498)	0.194(0.206)	0.756(0.139)	0.614(0.160)	0.838(0.019)	0.498(0.014)
DDQN	0.456(0.458)	0.168(0.196)	0.768(0.134)	0.650(0.216)	0.878(0.020)	0.528(0.017)
Rainbow	0.340(0.340)	0.156(0.160)	0.762(0.129)	0.602(0.207)	0.924(0.044)	0.524(0.022)
500 games						
DQN	0.430(0.430)	0.224(0.244)	0.742(0.136)	0.674(0.279)	0.700(0.015)	**0.534**(0.014)
DDQN	0.406(0.406)	0.218(0.228)	0.734(0.173)	0.626(0.213)	0.714(0.021)	0.508(0.026)
Rainbow	0.358(0.358)	0.190(0.196)	0.768(0.187)	0.656(0.207)	0.736(0.032)	0.496(0.029)
unlimited games						
DQN	0.300(0.300)	0.216(0.216)	0.752(0.119)	0.662(0.246)	0.562(0.034)	0.514(0.016)
DDQN	0.318(0.318)	0.258(0.258)	0.744(0.168)	0.628(0.134)	0.572(0.027)	0.480(0.024)
Rainbow	0.316(0.330)	**0.280**(0.280)	0.734(0.157)	**0.692**(0.157)	0.566(0.017)	0.514(0.014)

Table 9: Agent performance on **fixed map** games. Accuracies in percentage are shown in black. We also investigate the sufficient information bonus used in our agent proposed in Section 3.3.1, which are shown in blue.

Model	Location		Existence		Attribute	
	Train	Test	Train	Test	Train	Test
Human	–	1.000	–	1.000	–	0.750
Random	–	0.034	–	0.500	–	0.499
2 games						
DQN	0.990(0.990)	0.148(0.162)	1.000(0.779)	0.638(0.157)	1.000(0.039)	0.534(0.033)
DDQN	0.978(0.978)	0.146(0.152)	1.000(0.727)	0.602(0.158)	1.000(0.043)	**0.544**(0.032)
Rainbow	0.916(0.916)	0.178(0.178)	0.972(0.314)	0.602(0.136)	1.000(0.025)	0.512(0.021)
10 games						
DQN	0.818(0.818)	0.156(0.160)	0.898(0.607)	0.566(0.142)	1.000(0.056)	0.518(0.036)
DDQN	0.794(0.794)	0.142(0.154)	0.868(0.575)	0.606(0.153)	1.000(0.037)	0.500(0.033)
Rainbow	0.670(0.670)	0.144(0.170)	0.828(0.468)	0.586(0.128)	1.000(0.071)	0.530(0.018)
100 games						
DQN	0.550(0.550)	0.184(0.204)	0.758(0.230)	0.668(0.181)	0.878(0.021)	0.524(0.017)
DDQN	0.524(0.524)	0.188(0.204)	0.754(0.365)	0.662(0.205)	0.890(0.025)	**0.544**(0.019)
Rainbow	0.442(0.442)	0.174(0.184)	0.754(0.285)	0.654(0.190)	0.878(0.044)	0.504(0.032)
500 games						
DQN	0.430(0.430)	0.204(0.216)	0.752(0.162)	0.678(0.214)	0.678(0.019)	0.530(0.017)
DDQN	0.458(0.458)	0.222(0.246)	0.754(0.158)	0.656(0.188)	0.716(0.024)	0.486(0.023)
Rainbow	0.370(0.370)	0.172(0.178)	0.748(0.275)	0.678(0.191)	0.636(0.020)	0.494(0.017)
unlimited games						
DQN	0.316(0.316)	0.188(0.188)	0.728(0.213)	0.668(0.218)	0.812(0.055)	0.506(0.018)
DDQN	0.326(0.326)	0.206(0.206)	0.740(0.246)	**0.694**(0.196)	0.580(0.023)	0.482(0.017)
Rainbow	0.340(0.340)	**0.258**(0.258)	0.728(0.210)	0.686(0.193)	0.564(0.018)	0.470(0.017)

Table 10: Agent performance on **random map** games. Accuracies in percentage are shown in black. We also investigate the sufficient information bonus used in our agent proposed in Section 3.3.1, which are shown in blue.

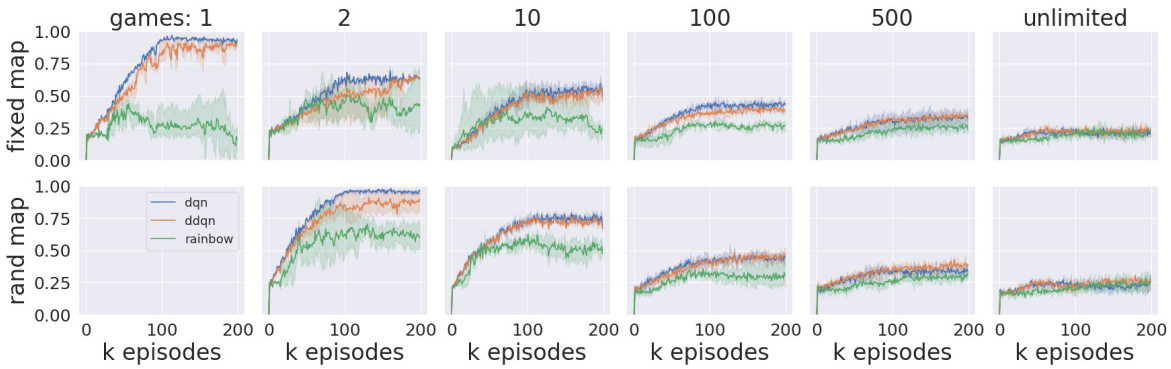

Figure 4: Training accuracy over episodes on **location** questions. Upper row: **fixed map**, 1/2/10/100/500/unlimited games; Lower row: **random map**, 2/10/100/500/unlimited games.

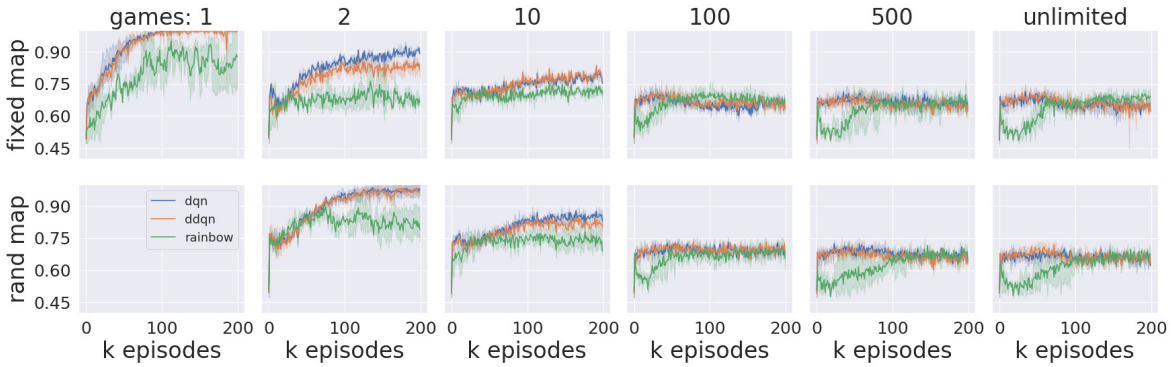

Figure 5: Training accuracy over episodes on **existence** questions. Upper row: **fixed map**, 1/2/10/100/500/unlimited games; Lower row: **random map**, 2/10/100/500/unlimited games.

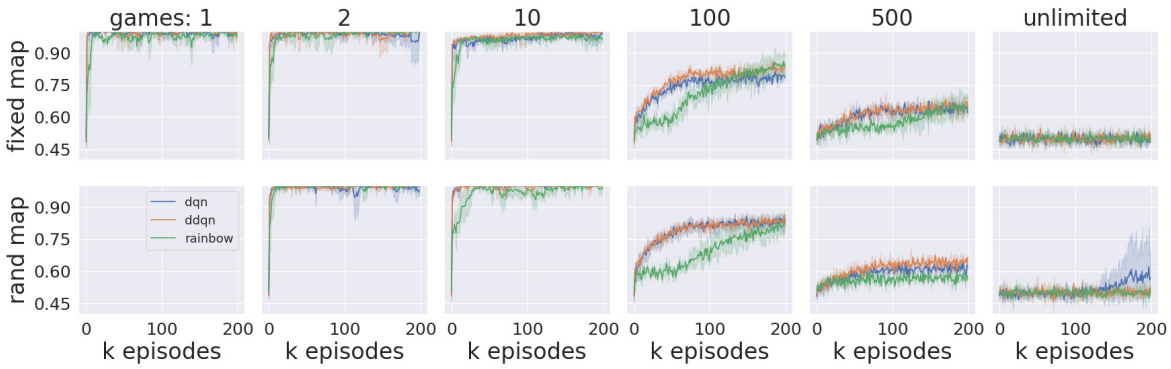

Figure 6: Training accuracy over episodes on **attribute** questions. Upper row: **fixed map**, 1/2/10/100/500/unlimited games; Lower row: **random map**, 2/10/100/500/unlimited games.