BABYAI: FIRST STEPS TOWARDS GROUNDED LANGUAGE LEARNING WITH A HUMAN IN THE LOOP

Anonymous Submission

ABSTRACT

Allowing humans to interactively train artificial agents to understand language instructions is desirable for both practical and scientific reasons, but given the poor data efficiency of the current learning methods, this goal may require substantial research efforts. Here, we introduce the BabyAI research platform to support investigations towards including humans in the loop for grounded language learning. The BabyAI platform comprises an extensible suite of 19 levels of increasing difficulty. The levels gradually lead the agent towards acquiring a combinatorially rich synthetic language which is a proper subset of English. The platform also provides a bot agent for the purpose of simulating a human teacher. We report baseline results and estimate the amount of human involvement that would be required to train a neural network-based agent on some of the BabyAI levels. We put forward strong evidence that current deep learning methods are not yet sufficiently sample efficient when it comes to learning a language with compositional properties.

1 INTRODUCTION

How can a human train an intelligent agent to understand natural language instructions? We believe that this research question is important from both technological and scientific perspectives. No matter how advanced AI technology becomes, human users may want to customize their intelligent helpers to be able to better understand their desires and needs. On the other hand, developmental psychology, cognitive science and linguistics study similar questions but applied to human children, and a synergy is possible between research in grounded language learning by computers and research in human language acquisition.

In this work, we take first steps towards studying grounded language learning with a human in the loop. In order to bootstrap this line of research, we present the BabyAI research platform, which includes a simulated human expert that teaches a neural learner. The current domain of BabyAI is a 2D gridworld and the synthetic instructions require the agent to navigate the world (including unlocking doors) and move objects to specified locations. BabyAI improves upon similar prior setups (Hermann et al., 2017; Chaplot et al., 2018; Yu et al., 2018) by supporting simulation of some of the essential aspects of the future human in the loop agent training: *curriculum learning* and *interactive teaching*.

The usefulness of curriculum learning for training machine learning models has been proven numerous times in the literature (Bengio et al., 2009; Kumar et al., 2010; Zaremba and Sutskever, 2015; Graves et al., 2016), and we believe that gradually increasing the difficulty of the task will likely be a key to efficient human-machine teaching, much like it is required for human-human teaching. To facilitate curriculum learning studies, BabyAI currently features 19 levels of increasing difficulty of the environment and the language.

Interactive teaching, i.e. teaching differently based on what the learner can currently achieve, is another key capability of a human teacher, and many effective interactive agent training methods, including DAGGER (Ross et al., 2011), TAMER (Warnell et al., 2017) and learning from human preferences (Wilson et al., 2012; Christiano et al., 2017) have already been proposed. To support interactive experiments, BabyAI provides a bot agent that can be used to provide new demonstrations on the fly and to give the learner advice on how to continue acting.

Arguably, the main obstacle to language learning with a human in the loop is the amount of data (and thus human-machine interactions) that would be required. Deep learning methods, used in the context of imitation learning or reinforcement learning paradigms, have been shown to be very effective in both simulated language learning settings (Mei et al., 2016; Hermann et al., 2017) and applications (Sutskever et al., 2014; Bahdanau et al., 2015; Wu et al., 2016), yet they typically require enormous amounts of data, either in terms of millions of reward function queries or hundreds of thousands of demonstrations. To show how our BabyAI platform can be used for data efficiency research, we perform several case studies on this topic. We measure the minimum number of samples that are required to solve several levels with imitation and reinforcement learning baselines. As first steps towards improving sample efficiency, we furthermore investigate how pretraining and interactive imitation learning can reduce the data demand.

The concrete contributions of this paper are thus two-fold. First, we contribute the BabyAI research platform for learning to perform language instructions with a simulated human in the loop. The platform already contains 19 levels and can be easily extended. Second, we establish baseline results for all levels and report data-efficiency results for a number of learning approaches. The platform and pretrained models will be available online. We hope that these will spur further research towards improving data efficiency of grounded language learning and teaching with a human in the loop.

2 Related Work

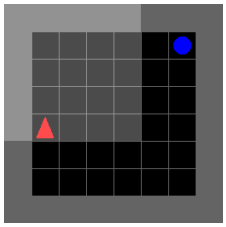
Many 2D and 3D environments with synthetic languages for studying language acquisition have recently been proposed (Hermann et al., 2017; Chaplot et al., 2018; Yu et al., 2018; Wu et al., 2018). The BabyAI platform draws inspiration from this prior work but is unique in combining a number of desirable features: (1) possibility of world state manipulation, missing in visually appealing 3D environments used by Hermann et al. (2017), Chaplot et al. (2018) and Wu et al. (2018), in which the agent can only navigate environment but cannot, for example move things around, (2) partial observability (missing in the gridworld of Bahdanau et al. (2018)) and (3) a systematic definition of the synthetic language. To elaborate on the last point, we note that, as opposed to using a handful of instruction templates, the Baby Language introduced here defines semantics for any utterance generated by a context-free grammar (see Section 3.2). This makes our language richer and more complete than those used in prior work. Most importantly, a unique property of the BabyAI platform is the availability of a simulated human expert that can be used to simulate human in the loop training, the focus of this paper.

There are multiple general-purpose 2D and 3D simulation frameworks out there which do not feature language, such as PycoLab (DeepMind, 2017), MazeBase (Sukhbaatar et al., 2015), Gazebo (Koenig and Howard, 2004), VizDoom (Kempka et al., 2016), DM-30 (Espeholt et al., 2018), and AI2-Thor (Kolve et al., 2017). We decided to use a gridworld environment because we believed this would allow better performance than was possible in a 3D environment, and that a 3D environment may add complexity that requires more development effort and detracts from the focus on grounded language learning. We chose to build the MiniGrid gridworld because we found available gridworld platforms to be insufficient for defining a compositional language.

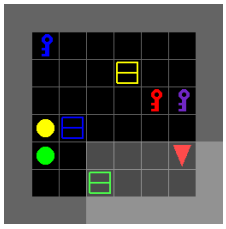
A related line of work is on general-purpose RL testbeds such as the Arcade Learning Environment (Bellemare et al., 2013), DM-30 (Espeholt et al., 2018), and MazeBase (Sukhbaatar et al., 2015). Unlike the aforementioned RL benchmarks, we assume a simulated human in the loop setting, in which all rewards (except intrinsic rewards) would have to be given by a human, and are therefore rather expensive to get. Under this assumption, imitation learning methods such as behavioral cloning, Searn (Daumé Iii et al., 2009), DAGGER (Ross et al., 2011) or maximum-entropy RL (Ziebart et al., 2008) are more appealing, as more learning can be achieved per unit of human input.

Similarly to the present work, studying data efficiency of deep learning methods was a goal of the bAbI tasks (Weston et al., 2016), which tested reasoning capabilities of the learning agent. Our work differs in both of the object of the study (grounded language with a simulated human in the loop) and in the method: instead of generating a fixed size dataset and measuring the performance we measure how much data a general-purpose model would require to get close-to-perfect performance.

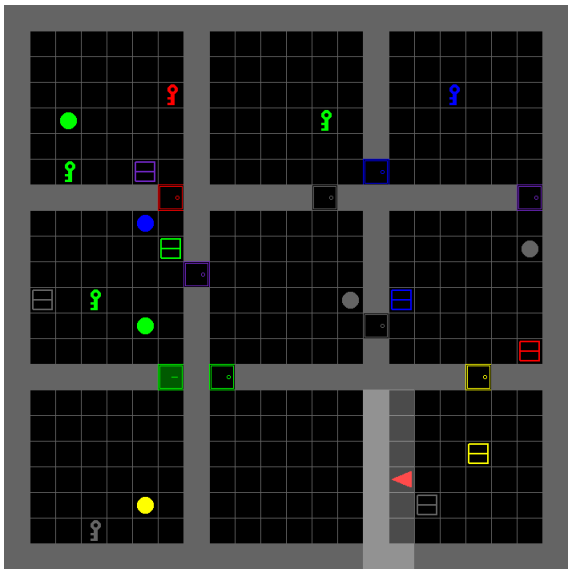
There has been much research on instruction following with natural language (Tellex et al., 2011; Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Mei et al., 2016; Williams et al., 2018) and



(a) GoToObj: "go to the blue ball"



(b) PutNextLocal: "put the blue key next to the green ball"



(c) BossLevel: "pick up the grey box behind you, then go to the grey key and open a door". Note that the green door near the bottom left needs to be unlocked with a green key, but this is not explicitly stated in the instruction.

Figure 1: Three BabyAI levels built using the MiniGrid environment. The red triangle represents the agent, and the light-grey shaded area represents its field of view (partial observation).

several datasets, such as e.g. SAIL (Macmahon et al., 2006; Chen and Mooney, 2011) and Room-to-Room (Anderson et al., 2018) are available for this purpose. We have however chosen to use a synthetic language for the BabyAI platform to have a fully controlled setting and to be able to generate as much data as needed.

Lastly, Wang et al. (2016) presented a system that was capable of learning language interactively from an actual human. We note that their system relied on substantial amounts of prior knowledge about the task, most importantly a task-specific executable formal language.

3 BabyAI Platform

The BabyAI platform that we present in this work comprises a fast and lightweight gridworld environment (MiniGrid) and a number of instruction-following tasks that we call *levels*, all formulated using subsets of a synthetic language (Baby Language). The platform also includes a bot that can solve all BabyAI levels and is an important component in defining a simulated teacher when evaluating human in the loop teaching methods. All the code is available online at `https://github.com/anonymous`.

3.1 MiniGrid Environment

Studies of data-efficiency are very computationally expensive (multiple runs are required for different amounts of data), hence, in our design of the environment, we have aimed for a minimalistic and efficient environment which still poses a considerable challenge for current general-purpose agent learning methods. We have implemented MiniGrid, a partially observable 2D gridworld environment. The environment is populated with various entities of different colors, such as the agent, balls, boxes, doors and keys (see Figure 1). Objects can be picked up, dropped and moved around by the agent, doors can be unlocked with keys matching their color. At each step, the agent receives a 7x7 representation of its field of view (the grid cells in front of it) as well as a Baby Language instruction (textual string).

The MiniGrid environment is fast and lightweight. Throughput of over 3000 frames per second is possible on a modern multi-core laptop, which makes experimentation quicker and more accessible. The environment is open source, available online, and supports integration with OpenAI Gym. For more details, see Appendix A.

3.2 BABY LANGUAGE

We have developed a synthetic Baby Language to give instructions to the agent as well as to automatically verify their execution. Baby Language is a comparatively small yet combinatorially rich subset of English that is designed to be easily understood by humans. In this language, the agent can be instructed to go to objects, pick up objects, open doors, and put objects next to other objects. The language can also express the conjunction of several such tasks, for example “put a red ball next to the green box after you open the door". The Backus-Naur Form (BNF) grammar for the language is presented in Figure 2 and some example instructions drawn from this language are shown in Figure 3. In order to keep the resulting instructions readable by humans, we have imposed some structural restrictions on this language: the *and* connector can only appear inside the *then* and *after* forms, and instructions can contain no more than one *then* or *after* word. The language is intentionally kept simple, but still exhibits interesting combinatorial properties, and contains 2.48×10^{19} possible instructions.

$$
\begin{aligned}
\langle\text{Sent}\rangle &\models \langle\text{Sent1}\rangle \mid \langle\text{Sent1}\rangle \text{ ',' then } \langle\text{Sent1}\rangle \mid \langle\text{Sent1}\rangle \text{ after you } \langle\text{Sent1}\rangle \\
\langle\text{Sent1}\rangle &\models \langle\text{Clause}\rangle \mid \langle\text{Clause}\rangle \text{ and } \langle\text{Clause}\rangle \\
\langle\text{Clause}\rangle &\models \text{go to } \langle\text{Descr}\rangle \mid \text{pick up } \langle\text{DescrNotDoor}\rangle \mid \text{open } \langle\text{DescrDoor}\rangle \mid \\
&\quad\ \text{put } \langle\text{DescrNotDoor}\rangle \text{ next to } \langle\text{Descr}\rangle \\
\langle\text{DescrDoor}\rangle &\models \langle\text{Article}\rangle\ \langle\text{Color}\rangle \text{ door } \langle\text{LocSpec}\rangle \\
\langle\text{DescrBall}\rangle &\models \langle\text{Article}\rangle\ \langle\text{Color}\rangle \text{ ball } \langle\text{LocSpec}\rangle \\
\langle\text{DescrBox}\rangle &\models \langle\text{Article}\rangle\ \langle\text{Color}\rangle \text{ box } \langle\text{LocSpec}\rangle \\
\langle\text{DescrKey}\rangle &\models \langle\text{Article}\rangle\ \langle\text{Color}\rangle \text{ key } \langle\text{LocSpec}\rangle \\
\langle\text{Descr}\rangle &\models \langle\text{DescrDoor}\rangle \mid \langle\text{DescrBall}\rangle \mid \langle\text{DescrBox}\rangle \mid \langle\text{DescrKey}\rangle \\
\langle\text{DescrNotDoor}\rangle &\models \langle\text{DescrBall}\rangle \mid \langle\text{DescrBox}\rangle \mid \langle\text{DescrKey}\rangle \\
\langle\text{LocSpec}\rangle &\models \epsilon \mid \text{on your left} \mid \text{on your right} \mid \text{in front of you} \mid \text{behind you} \\
\langle\text{Color}\rangle &\models \epsilon \mid \text{red} \mid \text{green} \mid \text{blue} \mid \text{purple} \mid \text{yellow} \mid \text{grey} \\
\langle\text{Article}\rangle &\models \text{the} \mid \text{a}
\end{aligned}
$$

Figure 2: BNF grammar productions for the Baby Language

go to the red ball

open the door on your left

put a ball next to the blue door

open the yellow door and go to the key behind you

put a ball next to a purple door after you put a blue box next to a grey box and pick up the purple box

Figure 3: Example Baby Language instructions

The BabyAI platform includes a *verifier* which serves to check if an agent performing a sequence of actions in a given environment has successfully completed a given instruction and achieved its goal or not. The descriptors in the language can refer to one or to multiple objects. Hence, if the agent is instructed to go to "a red door", it can execute this instruction by going to any of the red doors in the environment. The *then* and *after* connectors can be used to sequence subgoals. The *and* form implies that both subgoals must be completed, without ordering constraints. Importantly,

Baby Language instructions leave details about the execution implicit. An agent may have to find a key and unlock a door, or move obstacles out of the way to complete instructions, without this being stated explicitly.

3.3 BABYAI LEVELS

There is abundant evidence in the literature that using a curriculum may greatly facilitate learning complex tasks for neural architectures (Bengio et al., 2009; Kumar et al., 2010; Zaremba and Sutskever, 2015; Graves et al., 2016). To enable investigations of how a curriculum can help with data efficiency, we have produced a number of *levels* that require the understanding of only a limited of subset of Baby Language, and take place in environments of varying complexity. Formally, a level is a distribution of *missions*, where a mission is a combination of an instruction and an initial state of the environment. We have built levels by selecting a subset of *competencies* required for each level and implementing a generator of missions that can be solved by an agent that possesses only these competencies. Each competency is informally defined by specifying what an agent should be able to do:

- **Room Navigation (ROOM):** to navigate a 6x6 room
- **Ignoring Distracting Boxes (DISTR-BOX):** to navigate the environment even when there are multiple distracting grey box objects in it
- **Ignoring Distractors (DISTR):** same as DISTR-BOX, but distractor objects can be boxes, keys or balls of any color
- **Maze Navigation (MAZE):** to navigate a 3x3 maze of 6x6 rooms in which the rooms are randomly connected to each other with doors
- **Unblocking the Way (UNBLOCK):** to navigate the environment even when it requires moving the objects that are in the way
- **Unlocking Doors (UNLOCK):** to be able to find the key and unlock the door if the instruction requires this explicitly
- **Guessing to Unlock Doors (IMP-UNLOCK):** to guess that in order to execute instructions, the agent needs to identify the door that needs to be unlocked, find the respective key, unlock the door and proceed further with the execution
- **Go To Instructions (GOTO):** to understand "go to" instructions, e.g. "go to the red ball"
- **Open Instructions (OPEN):** to understand "open" instructions, e.g. "open the door on your left"
- **Pickup Instructions (PICKUP):** to understand "pick up" instructions, e.g. "pick up a box"
- **Put Instructions (PUT):** to understand "put" instructions, e.g. "put a ball next to the blue key"
- **Location Language (LOC):** to understand instructions in which objects are refered to by not only their shape and color but also by their location relative to the initial position of the agent, e.g. "go to the red ball in front of you"
- **Sequences of Commands (SEQ):** to understand composite instructions that require the agent to execute a sequence of instruction clauses, e.g. "put red ball next to the green box after you open the door"

Table 1 lists all current BabyAI levels together with the competencies required to solve them. These levels form a progression in terms of the competencies required to solve them, culminating with the BossLevel, which requires mastering all competencies. The definitions of competencies are informal and should be understood in the minimalistic sense, i.e. to test the ROOM competency we have built the GoToObj level where the agent needs to reach the only object in an empty room. Note that the GoToObj level does not require the GOTO competency, as this level can be solved without any language understanding, since there is only a single object in the room. However, solving the GoToLocal level, which instructs the agent to go to a specific object in the presence of multiple distractors, requires understanding GOTO instructions.

Table 1: BabyAI Levels and the required competencies

	ROOM	DISTR-BOX	DISTR	MAZE	UNBLOCK	UNLOCK	IMP-UNLOCK	GOTO	OPEN	PICKUP	PUT	LOC	SEQ
GoToObj	x												
GoToRedBallGrey	x	x											
GoToRedBall	x	x	x										
GoToLocal	x	x	x					x					
PutNextLocal	x	x	x								x		
PickUpLoc	x	x	x							x		x	
GoToObjMaze	x			x									
GoTo	x	x	x	x				x					
Pickup	x	x	x	x						x			
UnblockPickup	x	x	x	x	x					x			
Open	x	x	x	x					x				
Unlock	x	x	x	x		x			x				
PutNext	x	x	x	x							x		
Synth	x	x	x	x	x	x		x	x	x	x		
SynthLoc	x	x	x	x	x	x		x	x	x	x	x	
GoToSeq	x	x	x	x				x					x
SynthSeq	x	x	x	x	x	x		x	x	x	x	x	x
GoToImpUnlock	x	x	x	x			x	x					
BossLevel	x	x	x	x	x	x	x	x	x	x	x	x	x

3.4 THE BOT AGENT

The bot is a key ingredient intended to perform the role of a simulated human teacher. For any of the BabyAI levels, it can generate demonstrations or suggest actions for a given environment state. Whereas the BabyAI learner is meant to be generic and should scale to new and more complex tasks, the bot is engineered using knowledge of the tasks. This makes sense since the bot stands for the human in the loop, who is supposed to understand the environment, how to solve missions, and how to teach the baby learner. The bot has direct access to a tree representation of instructions, and so does not need to parse the Baby Language. Internally, it executes a stack machine in which instructions and subgoals are represented (more details can be found in Appendix B). The stack-based design allows the bot to interrupt what it is currently doing to achieve a new subgoal, and then resume the original task. For example, going to a given object will require exploring the environment to find that object.

The subgoals which the bot implements are:

- **Open:** Open a door that is in front of the agent.
- **Pickup:** Execute the pickup action.
- **Drop:** Execute the drop action.
- **GoToObj:** Go to an object matching a given (type, color) description.
- **GoNextTo:** Go to a cell adjacent to a given position.
- **GoToAdjPos:** Go next to a position adjacent to an object. This is necessary to implement the PutNext instruction.
- **Explore:** Uncover previously unseen parts of the environment. This is the most complex portion of the bot's internal logic.

All of the Baby Language instructions are decomposed into these internal subgoals which the bot knows how to solve. Many of these subgoals, during their execution, can also push new subgoals on the stack. A central part of the design of the bot is that it keeps track of the cells of the environment

which it has and has not seen. This is crucial to ensure that the bot can only use information which it could realistically have access to by exploring the environment. Exploration is implemented as part of the explore subgoal, which is recursive. For instance, exploring the environment may require opening doors, or moving objects that are in the way. Opening locked doors may in turn require finding a key, which may itself require exploration and moving obstructing objects. Another key component of the bot's design is a shortest path search routine. This is used to navigate to objects, to locate the closest door, or to navigate to the closest unexplored cell.

4 EXPERIMENTS

We assess the difficulty of BabyAI levels by training an imitation learning baseline for each level. Furthermore, we estimate how much data is required to solve some of the simpler levels and study to which extent the data demands can be reduced by using basic curriculum learning and interactive teaching methods. All the code that we use for the experiments, as well as containerized pretrained models, is available online.

4.1 SETUP

The BabyAI platform provides by default a 7x7x3 symbolic observation x_t (a partial and local egocentric view of the state of the environment) and a variable length instruction c as inputs at each step. We use a basic model consisting of standard components to predict the next action a based on x and c. In particular, we use a GRU (Cho et al., 2014) to encode the instruction and a convolutional network with two batch-normalized (Ioffe and Szegedy, 2015) FiLM (Perez et al., 2017) layers to jointly process the observation and the instruction. An LSTM (Hochreiter and Schmidhuber, 1997) memory is used to integrate representations produced by the FiLM module at each step. Our model is thus similar to the gated-attention model used by Chaplot et al. (2018), inasmuch as gated attention is equivalent to using FiLM without biases and only at the output layer.

Previous work has shown that attention can significantly the improve performance of agents in grounded language learning experiments Chaplot et al. (2018). We have chosen to use FiLM over a gated attention mechanism because the FiLM mechanism seems like a more flexible alternative, which can not only rescale convolutional feature maps, but also conditionally re-normalize them.

We have used two versions of our model, to which we will refer as the Large model and the Small model. In the Large model, the memory LSTM has 2048 units and the instruction GRU is bidirectional and has 256 units. Furthermore, an attention mechanism (Bahdanau et al., 2015) is used to focus on the relevant states of the GRU. The Small model uses a smaller memory of 128 units and encodes the instruction with a unidirectional GRU and no attention mechanism.

In all our experiments, we used the Adam optimizer (Kingma and Ba, 2015) with the hyperparameters $\alpha = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-5}$. In our imitation learning (IL) experiments, we truncated the backpropagation through time at 20 steps for the Small model and at 80 steps for the Large model. For our reinforcement learning experiments, we used the Proximal Policy Optimization (PPO, Schulman et al., 2017) algorithm with parallelized data collection. Namely, we performed 4 epochs of PPO using 64 rollouts of length 40 collected with multiple processes. We gave a non-zero reward to the agent only when it fully completed the mission, and the magnitude of the reward was $1 - 0.9n/n_{max}$, where n is the length of the successful episode and n_{max} is the maximum number of steps that we allowed for completing the episode, different for each mission. The reward future returns were discounted without a factor $\gamma = 0.99$. For generalized advantage estimation (Schulman et al., 2015) in PPO we used $\lambda = 0.99$.

In all our experiments we reported the success rate, defined as the ratio of missions of the level that the agent was able to accomplish within n_{max} steps.

Running the experiments outlined in this section required between 20 and 50 GPUs during two weeks. At least as much computing was required for preliminary investigations.

Table 2: Baseline imitation learning results for all BabyAI levels. Each model was trained with one million demonstrations from the respective level. For reference, we also list the mean demonstration length for each level.

Model	Success Rate (%)	Mean Demo Length
GoToImpUnlock	79.30	151
BossLevel	80.86	114
SynthSeq	85.16	102
GoToSeq	91.41	94
SynthLoc	96.09	58
Unlock	96.29	124
PutNext	97.66	122
Synth	97.85	69
Pickup	99.41	71
GoTo	99.61	70
UnblockPickup	99.80	72
GoToLocal	100.00	6.5
GoToObj	100.00	5.7
GoToObjMaze	100.00	91
GoToRedBallGrey	100.00	6.8
GoToRedBall	100.00	6.5
Open	100.00	43
PickupLoc	100.00	8.6
PutNextLocal	100.00	13.9

4.2 Baseline Results

To obtain baseline results for all BabyAI levels, we have trained the Large model (see Section 4.1) with imitation learning using one million demonstration episodes for each level. The demonstrations were generated using the bot described in section 3.4. The models were trained for approximately a week. Table 2 reports the final success rate on a validation set of 512 episodes. All of the single-room levels are solved with a success rate of 100.0%. As a general rule, levels for which longer demonstrations tend to be more difficult to solve.

Using 1M demonstrations for levels as simple as GoToRedBall is very inefficient and hardly ever compatible with the long-term goal of enabling human teaching. The BabyAI platform is meant to support studies of how neural agents can learn with less data. To bootstrap such studies we have computed baseline data efficiencies for imitation learning and reinforcement learning approaches to solving BabyAI levels. We say an agent solves a level if it reaches a success rate of at least 99%. We define the data efficiency as the minimum number of demonstrations or RL episodes required to train an agent solve a given level. To estimate the data efficiency for imitation learning, we have tried training models with different numbers of demonstrations starting from one million and dividing each time by $\sqrt{2}$. Each model was trained for $2 \cdot T^L_{min}$ parameter updates, where T^L_{min} is the number of parameter updates that was required for getting the target 99% performance with 1M demonstrations for the level L. For each level we find the minimum number of demonstrations k from our $\sqrt{2}$ grid for which the 99% threshold was crossed in at least in 1 run out of 3. We can then be sure that the minimum number of demonstrations lies somewhere in the $\left[k/\sqrt{2}; k\right]$ bracket. The results for a subset of levels are reported in Table 3 (see "IL from Bot" column). In the same table (column "RL") we report the number of episodes that were required for reinforcement learning to solve each of these levels, and as expected, the data efficiency of RL is substantially worse than that of IL (anywhere between 4 to 8 times in these experiments).

To analyze how much the data efficiency of IL depends on the source of demonstrations, we experimented with generating demonstrations with agents that were trained with RL for the previous experiments. The results are reported in the "IL from RL" column in Table 5. Interestingly, we found that the demonstrations produced by such an agent are easier for the learner to imitate (for example, for GoToLocal 70.7K demonstrations were sufficient to imitate the RL expert as opposed to 177K needed to imitate the bot). This can explained by the fact that the RL expert has the same neural network architecture as the learner.

Table 3: The data efficiency of imitation learning and reinforcement learning as the number of demonstrations (episodes) required to solve each level. All numbers are thousands. For RL experiments we report the minimum and the maximum data efficiency observed in several runs. For the baseline imitation learning results we report a $\left[k/\sqrt{2}; k\right]$ bracket, see Section 4 for details.

Level	IL from Bot	RL
GoToRedBallGrey	5.7 - 8	377 - 379
GoToRedBall	44.2 - 62.5	453 - 470
GoToLocal	125.2 - 177	1167 - 1320
PickupLoc	250 - 354	2591 - 2608
PutNextLocal	354 - 500	1875 - 2587
GoTo	250 - 354	1057 - 2177

Table 4: The data efficiency results for pretraining experiments. For each pair of base levels and target levels that we have tried, we report how many demonstrations were required, as well as the baseline number of demonstrations required for training from scratch. In both cases we report a $\left[k/\sqrt{2}; k\right]$ range, see Section 4 for details. Note how choosing the right base levels (e.g. GoToLocal and GoToObjMaze) is crucial for pretraining to be helpful.

Base Levels	Target Level	With Pretraining	Without Pretraining
GoToLocal	GoTo	250 - 354	250 - 354
GoToObjMaze	GoTo	354 - 500	250 - 354
GoToLocal and GoToObjMaze	GoTo	88.4 - 125	250 - 354
GoToLocal	PickupLoc	177 - 250	250 - 354
GoToLocal	PutNextLocal	250 - 354	354 - 500

4.3 Curriculum Learning

To demonstrate how curriculum learning research can be done using the BabyAI platform, we perform a number of basic pretraining experiments. In particular, we select 5 combinations of base levels and a target level and study if pretraining on base levels can help the agent master the target level with less demonstrations. The results are reported in Table 4. Pretraining was most helpful when GoToLocal and GoToObjMaze were used as the base levels and GoTo was used as target level, reducing the number of demonstrations required to solve GoTo from 354K to 125K. In other cases, e.g. when only GoToObjMaze was used as the base level, we have not found pretraining to be clearly beneficial. We find this counter-intuitive result interesting, as it shows how current deep learning methods often can not take the full advantage of available curriculums.

4.4 Interactive Learning

Lastly, we perform an example case study of how data efficiency can be improved by interactively providing more informative examples to the agent based on what it has already learned. We experiment with an iterative algorithm for adaptively growing the agent's training set. In particular, we start with 5000 base demonstrations, and at each iteration we increase the dataset size by the factor of 1.2 by providing bot demonstrations for missions which the agent failed. After each dataset increase we train a new agent from scratch. We then report the size of the training set for which the agent's performance has surpassed the 99% threshold. We repeat such an experiment 4 times for levels GoToRedBallGrey, GoToRedBall and GoToLocal and report the maximum and the minimum data efficiency for this approach, which we call interactive imitation learning, in Table 5. We have observed substantial improvement on the vanilla IL in some runs (e.g. 111K vs 192K for GoToLocal), but it should be noted the variance of interactive imitation learning results was rather high.

5 Conclusion & Future Work

We present the BabyAI research platform to study language learning with a human in the loop. The platform includes 19 levels of increasing difficulty, based on a decomposition of tasks into a set of basic competencies. Solving the levels requires understanding the Baby Language, a subset of

Table 5: The data efficiency of imitation learning (IL) from an RL-pretrained expert and interactive imitation learning defined as the number of demonstrations required to solve each level. All numbers are thousands. For interactive IL we report the minimum and the maximum data efficiency observed in several runs. For the baseline imitation learning results we report a $\left[k/\sqrt{2}; k\right]$ range, see Section 4 for details.

Level	IL from Bot	IL from RL Expert	Interactive IL from Bot
GoToRedBallGrey	5.7 - 8	1.4	3 - 4.3
GoToRedBall	44.2 - 62.5	50	55 - 66
GoToLocal	125.2 - 177	70.7	111 - 192

English with a formally defined grammar which exhibits compositional properties. The language is minimalistic and the levels seem simple, but empirically we have found them quite challenging to solve. The platform is open source and extensible, meaning new levels and language concepts can be integrated easily.

The results in Section 4 suggest that current imitation learning and reinforcement learning methods scale and generalize poorly when it comes to learning tasks with a compositional structure. Hundreds of thousands of demonstrations are needed to learn tasks which seem trivial by human standards. Methods such as curriculum learning and interactive learning can provide measurable improvements in terms of data efficiency, but, in order for learning with an actual human in the loop to become realistic, an improvement of at least three orders of magnitude is required.

An obvious direction of future research to find strategies to improve data efficiency of language learning. Tackling this challenge will likely require new models and new teaching methods. Approaches that involve an explicit notion of modularity and subroutines, such as Neural Module Networks (Andreas et al., 2016) or Neural Programming Interpreters (Reed and de Freitas, 2015), seem like a promising direction. It is our hope that the BabyAI platform can serve as a challenge and a benchmark for the data efficiency of language learning for years to come.

ACKNOWLEDGEMENTS

We thank Tristan Deleu and Saizheng Zhang for useful discussions. This research was enabled in part by support provided by Compute Canada (www.computecanada.ca), NSERC and Canada Research Chairs. We also thank Nvidia for donating NVIDIA DGX-1 used for this research.

REFERENCES

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and Hengel, A. v. d. (2018). Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. (2016). Neural Module Networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Artzi, Y. and Zettlemoyer, L. (2013). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*, 1:49–62.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of the ICLR 2015*.

Bahdanau, D., Hill, F., Leike, J., Hughes, E., Kohli, P., and Grefenstette, E. (2018). Learning to Follow Language Instructions with Adversarial Reward Induction. *arXiv:1806.01946 [cs]*. arXiv: 1806.01946.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279. arXiv: 1207.4708.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R. (2018). Gated-Attention Architectures for Task-Oriented Language Grounding. In *Proceedings of 32nd AAAI Conference on Artificial Intelligence*.

Chen, D. L. and Mooney, R. J. (2011). Learning to Interpret Natural Language Navigation Instructions from Observations. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 859–865.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems 30*. arXiv: 1706.03741.

Daumé Iii, H., Langford, J., and Marcu, D. (2009). Search-based structured prediction. *Machine learning*, 75(3):297–325.

DeepMind (2017). PycoLab.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *Proceedings of the 22nd international conference on Machine learning*. arXiv: 1802.01561.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., Wainwright, M., Apps, C., Hassabis, D., and Blunsom, P. (2017). Grounded Language Learning in a Simulated 3d World. *arXiv:1706.06551 [cs, stat]*.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaskowski, W. (2016). ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *CIG*, pages 1–8. IEEE.

Kingma, D. P. and Ba, J. (2015). A method for stochastic optimization. In *2015 International Conference on Learning Representation*.

Koenig, N. and Howard, A. (2004). Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, Sendai, Japan.

Kolve, E., Mottaghi, R., Gordon, D., Zhu, Y., Gupta, A., and Farhadi, A. (2017). AI2-THOR: An Interactive 3d Environment for Visual AI. *CoRR*, abs/1712.05474.

Kumar, M. P., Packer, B., and Koller, D. (2010). Self-Paced Learning for Latent Variable Models. In *Advances in Neural Information Processing Systems 23*, pages 1189–1197. Curran Associates, Inc.

Macmahon, M., Stankiewicz, B., and Kuipers, B. (2006). Walk the Talk: Connecting Language, Knowledge, Action in Route Instructions. In *In Proc. of the Nat. Conf. on Artificial Intelligence (AAAI*, pages 1475–1482.

Mei, H., Bansal, M., and Walter, M. R. (2016). Listen, Attend, and Walk: Neural Mapping of Navigational Instructions to Action Sequences. In *Proceedings of the 2016 AAAI Conference on Artificial Intelligence*.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. (2017). FiLM: Visual Reasoning with a General Conditioning Layer. In *In Proceedings of the 2017 AAAI Conference on Artificial Intelligence*.

Reed, S. and de Freitas, N. (2015). Neural Programmer-Interpreters. In *2016 International Conference on Learning Representations*. arXiv: 1511.06279.

Ross, S., Gordon, G., and Bagnell, D. (2011). A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *PMLR*, pages 627–635.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015). High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *Advances in Neural Information Processing Systems 30*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*. arXiv: 1707.06347.

Sukhbaatar, S., Szlam, A., Synnaeve, G., Chintala, S., and Fergus, R. (2015). MazeBase: A Sandbox for Learning from Games. *arXiv:1511.07401 [cs]*. arXiv: 1511.07401.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112.

Tellex, S., Kollar, T., Dickerson, S., Walter, M. R., Banerjee, A. G., Teller, S., and Roy, N. (2011). Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Wang, S. I., Liang, P., and Manning, C. D. (2016). Learning Language Games through Interaction. In *Proceedings Of the 54th Annual Meeting of the Association for Computational Linguistics*. arXiv: 1606.02447.

Warnell, G., Waytowich, N., Lawhern, V., and Stone, P. (2017). Deep TAMER: Interactive Agent Shaping in High-Dimensional State Spaces. In *Proceedings of 32nd AAAI Conference on Artificial Intelligence*. arXiv: 1709.10163.

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., and Mikolov, T. (2016). Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks.

Williams, E. C., Gopalan, N., Rhee, M., and Tellex, S. (2018). Learning to Parse Natural Language to Grounded Reward Functions with Weak Supervision. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 1–7.

Wilson, A., Fern, A., and Tadepalli, P. (2012). A Bayesian Approach for Policy Learning from Trajectory Preference Queries. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1133–1141. Curran Associates, Inc.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., and others (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*.

Wu, Y., Wu, Y., Gkioxari, G., and Tian, Y. (2018). Building Generalizable Agents with a Realistic and Rich 3d Environment. *arXiv:1801.02209 [cs]*. arXiv: 1801.02209.

Yu, H., Zhang, H., and Xu, W. (2018). Interactive Grounded Language Acquisition and Generalization in 2d Environment. In *ICLR*.

Zaremba, W. and Sutskever, I. (2015). Learning to Execute. In *2015 International Conference on Learning Representations*. arXiv: 1410.4615.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). Maximum Entropy Inverse Reinforcement Learning. In *Proc. AAAI*, pages 1433–1438.

A MINIGRID ENVIRONMENTS FOR OPENAI GYM

The environments used for this research are built on top of MiniGrid, which is an open source grid-world package. This package includes a family of reinforcement learning environments compatible with the OpenAI Gym framework. Many of these environments are parameterizable so that the difficulty of tasks can be adjusted (e.g. the size of rooms is often adjustable).

A.1 THE WORLD

In MiniGrid, the world is a grid of size NxN. Each tile in the grid contains exactly zero or one object, and the agent can only be on an empty tile or on a tile containing an open door. The possible object types are wall, door, key, ball, box and goal. Each object has an associated discrete color, which can be one of red, green, blue, purple, yellow and grey. By default, walls are always grey and goal squares are always green.

A.2 REWARD FUNCTION

Rewards are sparse for all MiniGrid environments. Each environment has an associated time step limit. The agent receives a positive reward if it succeeds in satisfying an environment's success criterion within the time step limit, otherwise zero. The formula for calculating positive sparse rewards is $1 - 0.9 * (step_count/max_steps)$. That is, rewards are always between zero and one, and the quicker the agent can successfully complete an episode, the closer to 1 the reward will be. The max_steps parameter is different for each mission, and varies depending on the size of the environment (larger environments having a higher time step limit) and the length of the instruction (more time steps are allowed for longer instructions).

A.3 ACTION SPACE

There are seven actions in MiniGrid: turn left, turn right, move forward, pick up an object, drop an object, toggle and done. The agent can use the turn left and turn right action to rotate and face one of 4 possible directions (north, south, east, west). The move forward action makes the agent move from its current tile onto the tile in the direction it is currently facing, provided there is nothing on that tile, or that the tile contains an open door. The agent can open doors if they are right in front of it by using the toggle action.

A.4 OBSERVATION SPACE

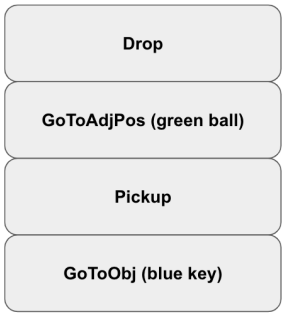
Observations in MiniGrid are partial and egocentric. By default, the agent sees a square of 7x7 tiles in the direction it is facing. These include the tile the agent is standing on. The agent cannot see through walls or closed doors. The observations are provided as a tensor of shape 7x7x3. However, note that these are not RGB images. Each tile is encoded using 3 integer values: one describing the type of object contained in the cell, one describing its color, and a flag indicating whether doors are open or closed. This compact encoding was chosen for space efficiency and to enable faster training. The fully observable RGB image view of the environments shown in this paper is provided for human viewing.

B BOT IMPLEMENTATION DETAILS

B.1 TRANSLATION OF INSTRUCTIONS INTO SUBGOALS

The bot has access to a representation of the instructions for each environment. These instructions are decomposed into subgoals that are added to a stack. In Figure 4 we show the stacks corresponding to the examples in Figure 1. The stacks are illustrated in bottom to top order, that is, the lowest subgoal in the illustration is to be executed first.

(a) GoToObj: "go to the blue ball"



(b) PutNextLocal: "put the blue key next to the green ball"

(c) BossLevel: "pick up the grey box behind you, then go to the grey key and open a door".

Figure 4: Examples of stacks corresponding to three different instructions

B.2 Processing of subgoals

Once instructions for a task are translated into the initial stack of subgoals, the bot starts by processing the first subgoal. Each subgoal is processed independently, and can either lead to more subgoals being added to the stack, or to an action being taken. When an action is taken, the state of the bot in the environment changes, and its visibility mask is populated with all the new observed cells and objects, if any. The visibility mask is essential when looking for objects and paths towards cells, because it keeps track of what the bot has seen so far. Once a subgoal is marked as completed, it is removed from the stack, and the bot starts processing the next subgoal in the stack.

A complete description of how the subgoals are handled is depicted in Figures 5, 6, 7, 8, 9, and 10. We make use of the additional "Unblock" subgoal, that we describe in Figure 11, in order to simplify the description of the main subgoals.

In the diagrams, we use the term "forward cell" to refer to the grid cell that the agent is facing. We say that a path from X to Y contains blockers if there are objects that need to be moved in order for the agent to be able to navigate from X to Y. A "clear path" is a path without blockers.

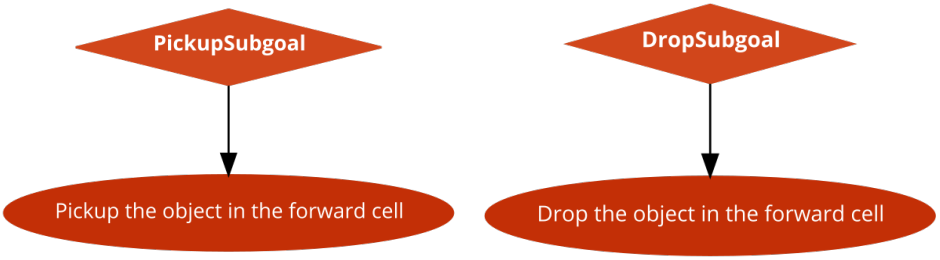


Figure 5: Processing of the Pickup and Drop subgoals

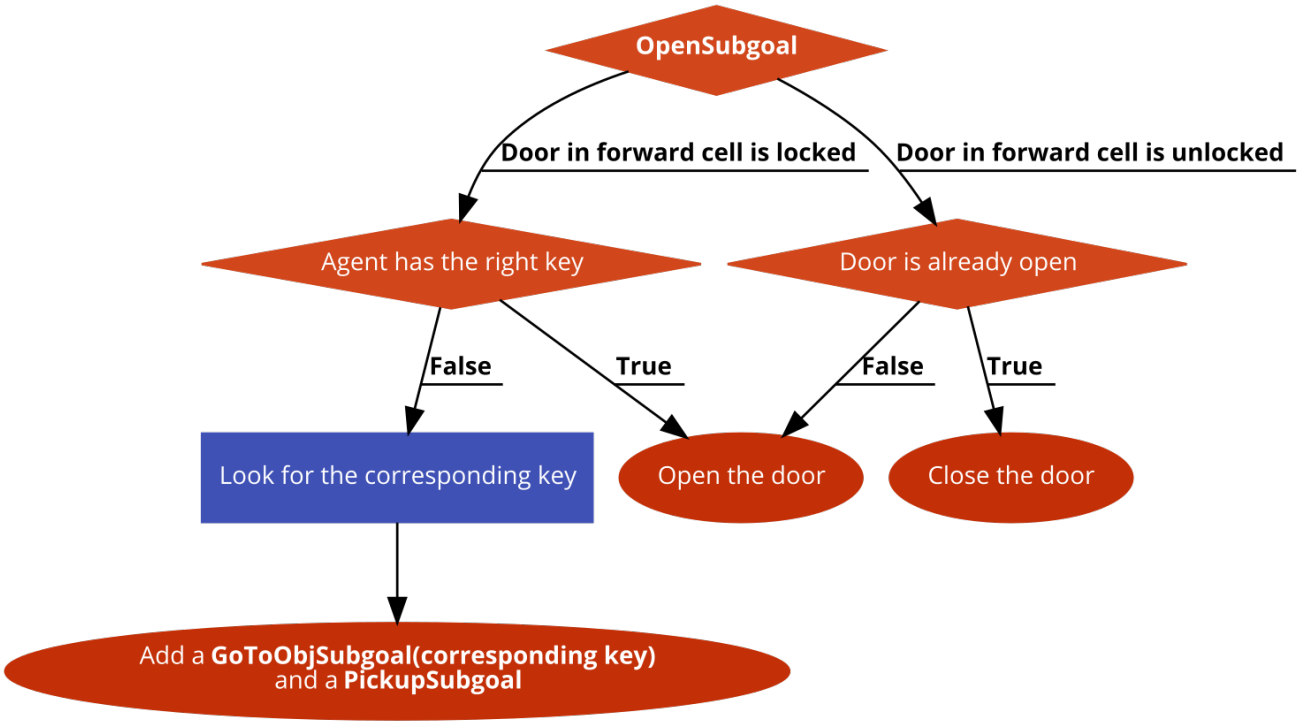


Figure 6: Processing of the Open subgoal

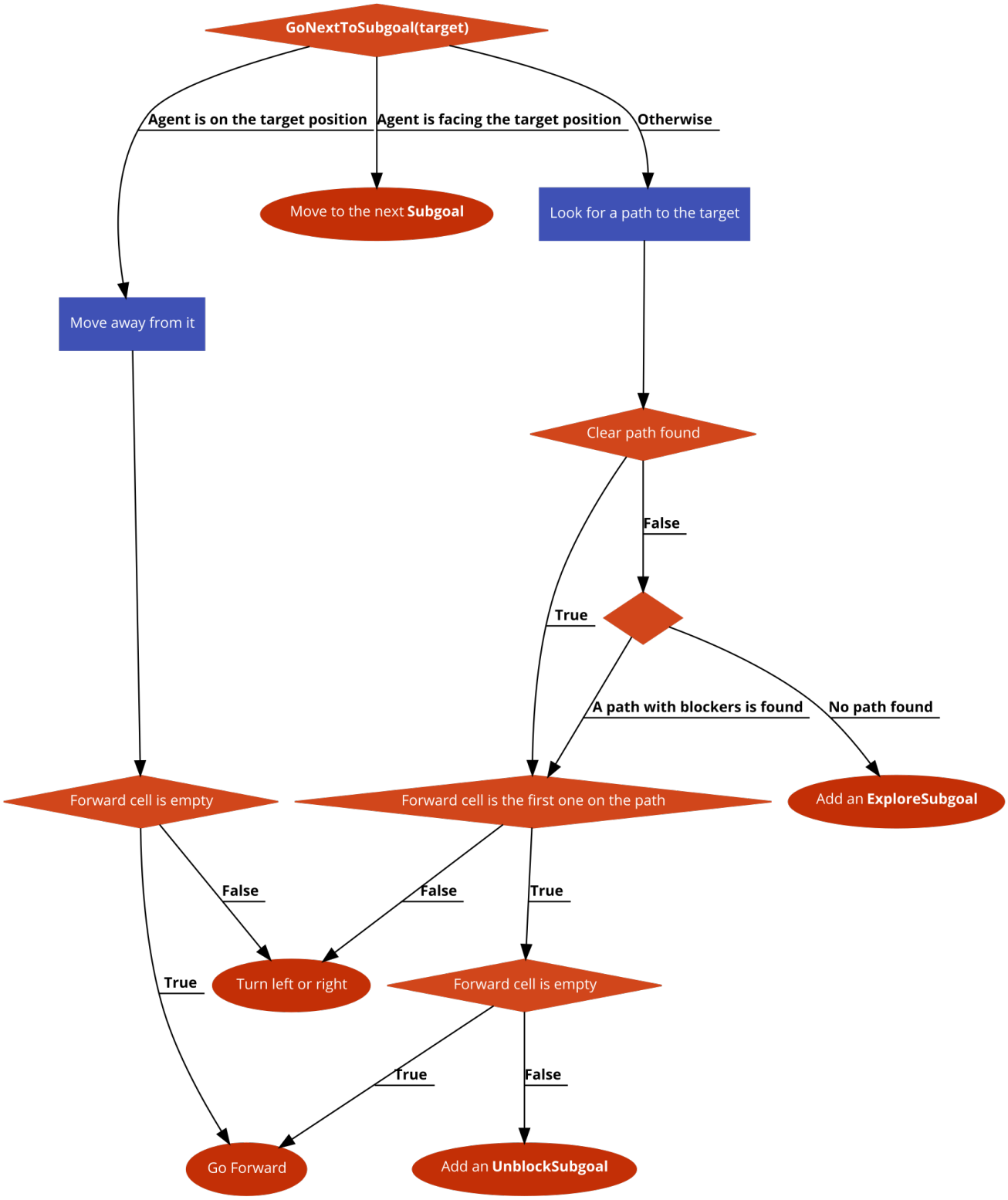


Figure 7: Processing of the GoNextTo subgoal

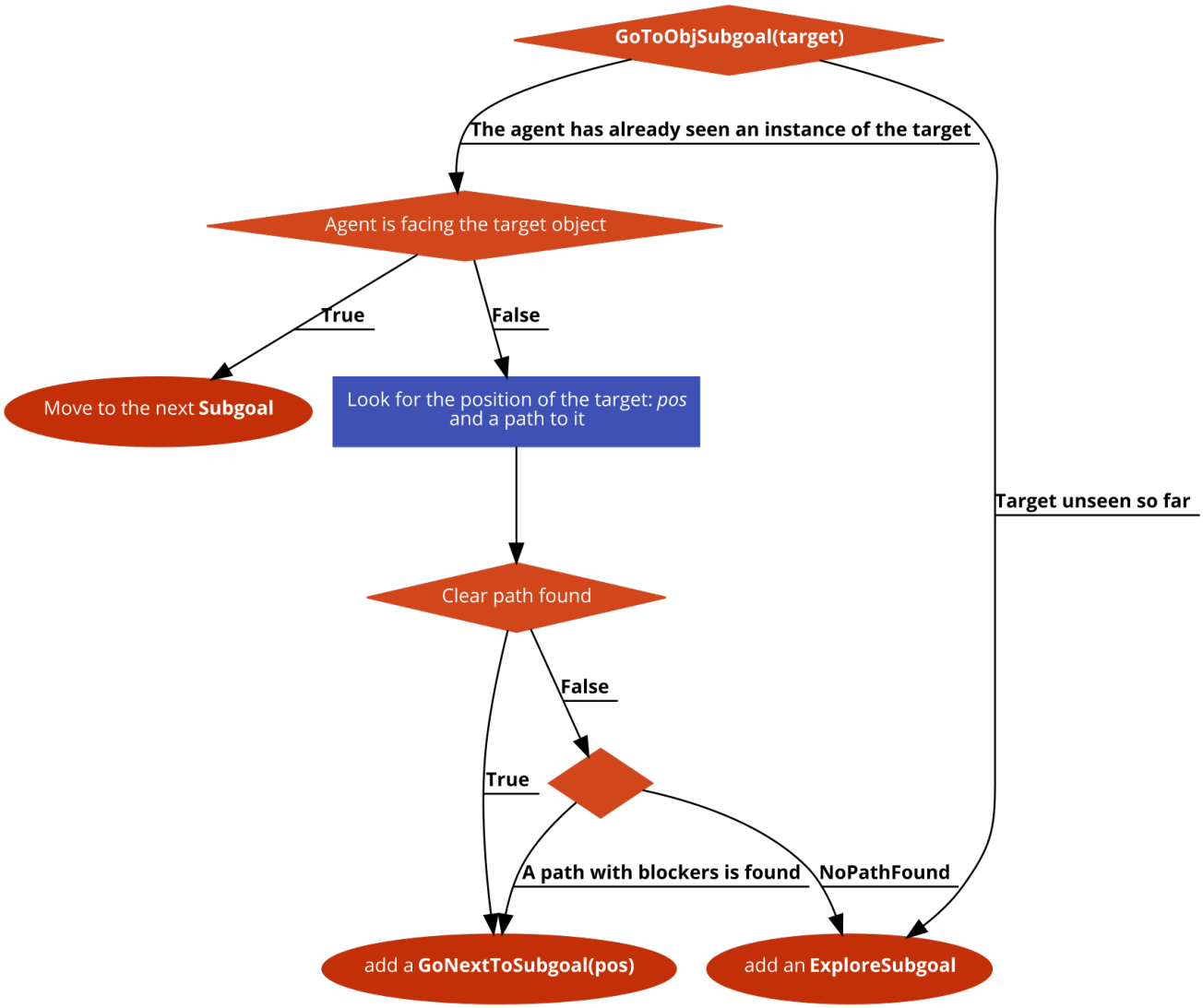


Figure 8: Processing of the GoToObj subgoal

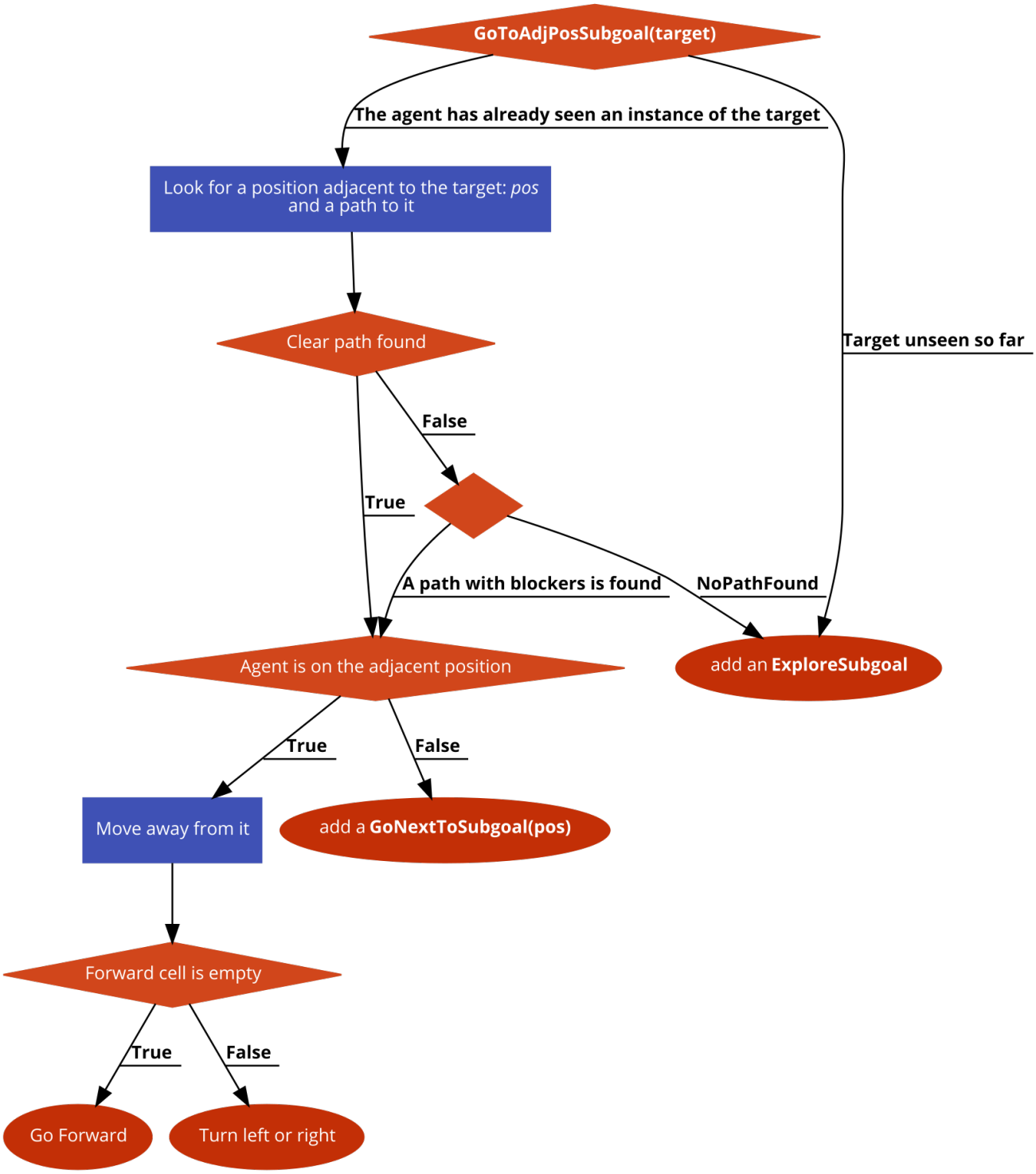


Figure 9: Processing of the GoToAdjPos subgoal

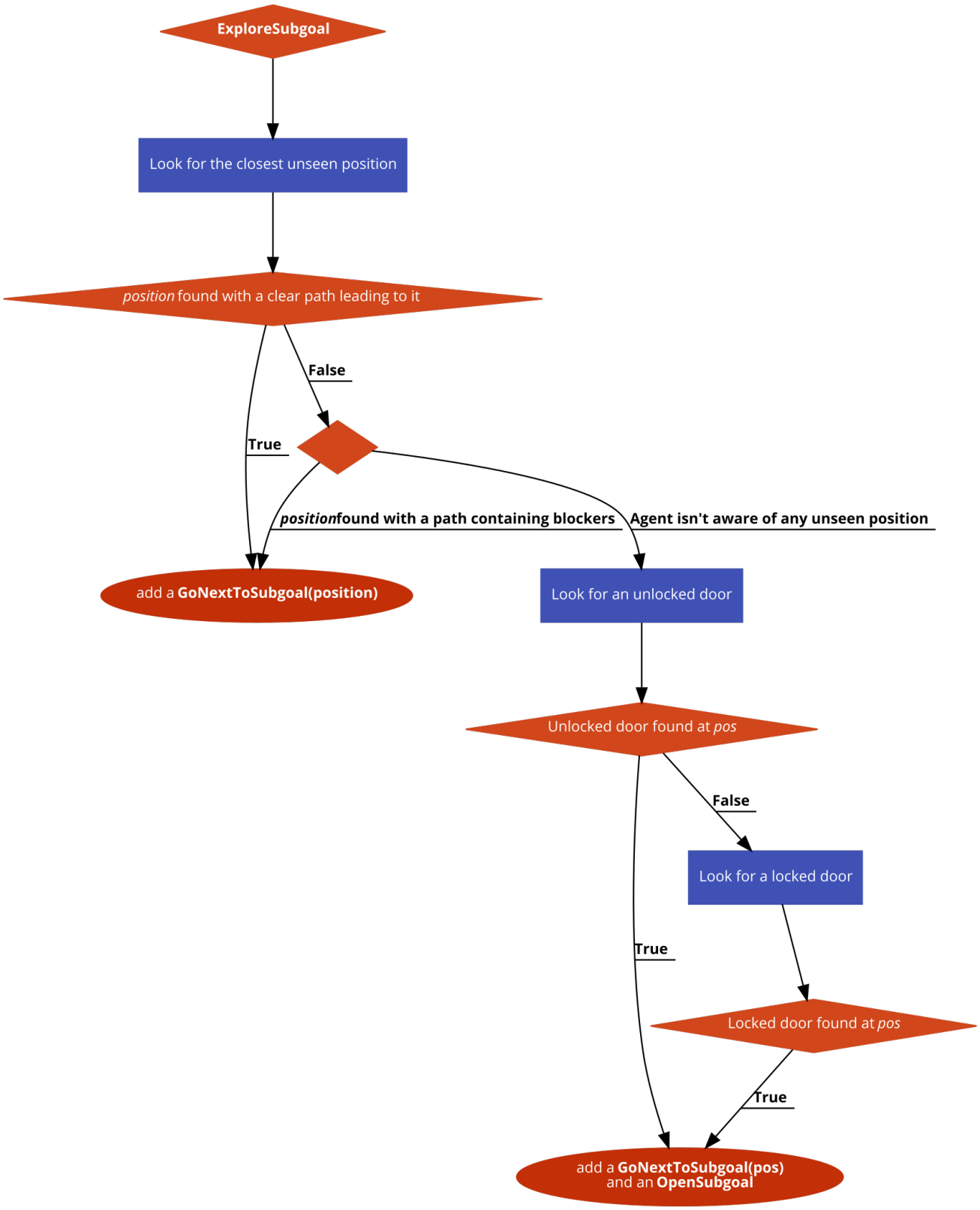


Figure 10: Processing of the Explore subgoal

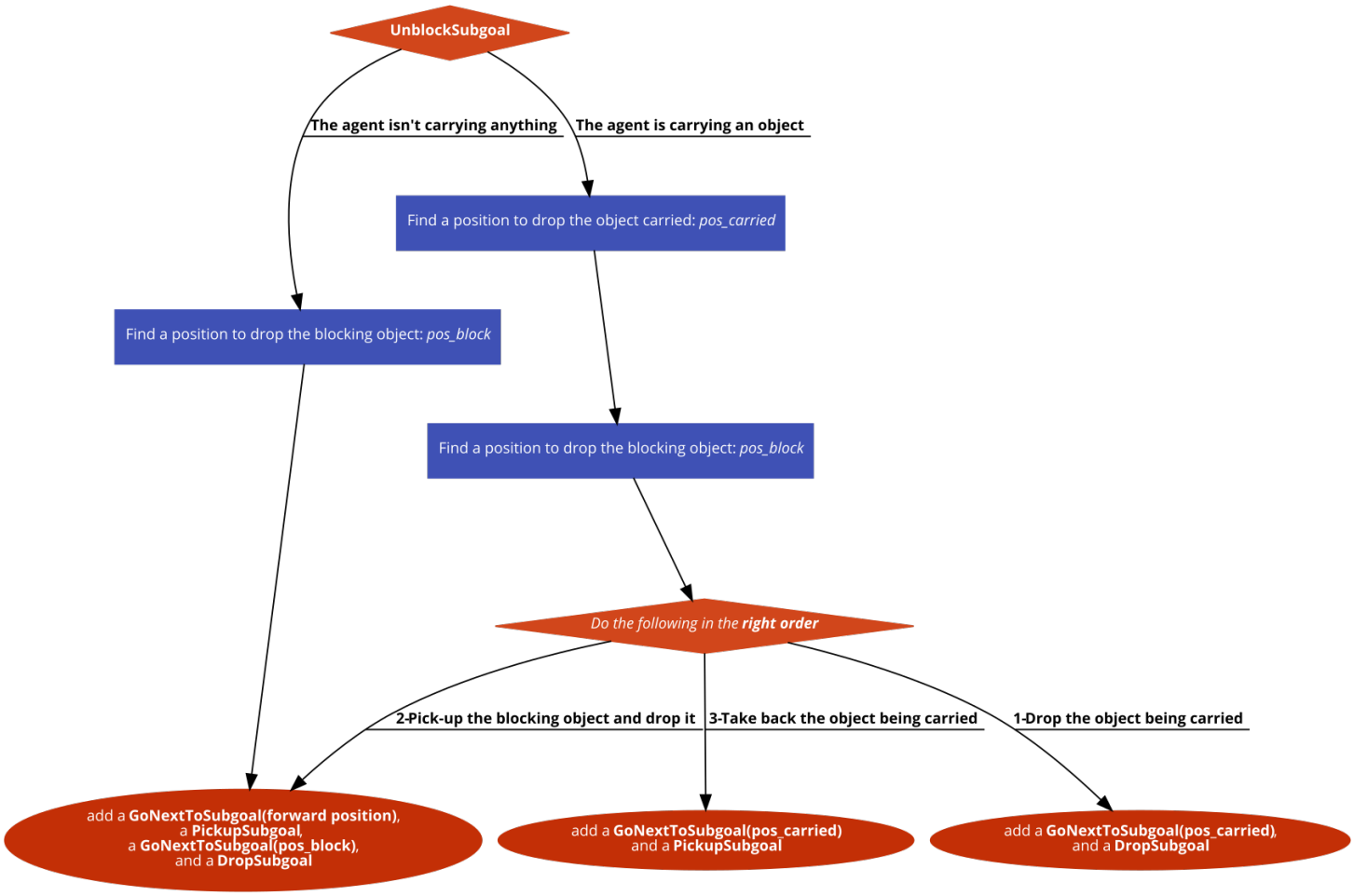


Figure 11: Processing of the Unblock subgoal