

Control of Memory, Active Perception, and Action in Minecraft

Junhyuk Oh
 Valliappa Chockalingam
 Satinder Singh
 Honglak Lee

Computer Science & Engineering, University of Michigan

JUNHYUK@UMICH.EDU
 VALLI@UMICH.EDU
 BAVEJA@UMICH.EDU
 HONGLAK@UMICH.EDU

Abstract

In this paper, we introduce a new set of reinforcement learning (RL) tasks in Minecraft (a flexible 3D world). We then use these tasks to systematically compare and contrast existing deep reinforcement learning (DRL) architectures with our new memory-based DRL architectures. These tasks are designed to emphasize, in a controllable manner, issues that pose challenges for RL methods including partial observability (due to first-person visual observations), delayed rewards, high-dimensional visual observations, and the need to use active perception in a correct manner so as to perform well in the tasks. While these tasks are conceptually simple to describe, by virtue of having all of these challenges simultaneously they are difficult for current DRL architectures. Additionally, we evaluate the generalization performance of the architectures on environments not used during training. The experimental results show that our new architectures generalize to unseen environments better than existing DRL architectures.

1. Introduction

Deep learning approaches (surveyed in LeCun et al., 2015; Schmidhuber, 2015) have made advances in many low-level perceptual supervised learning problems (Krizhevsky et al., 2012; Girshick et al., 2014; Simonyan & Zisserman, 2015). This success has been extended to reinforcement learning (RL) problems that involve visual perception. For example, the Deep Q-Network (DQN) (Mnih et al., 2015) architecture has been shown to successfully learn to play many Atari 2600 games in the Arcade Learning Environment (ALE) benchmark (Bellemare et al., 2013) by learning visual features useful for control directly from raw pixels using Q-Learning (Watkins & Dayan, 1992).

Proceedings of the 33rd International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48. Copyright 2016 by the author(s).

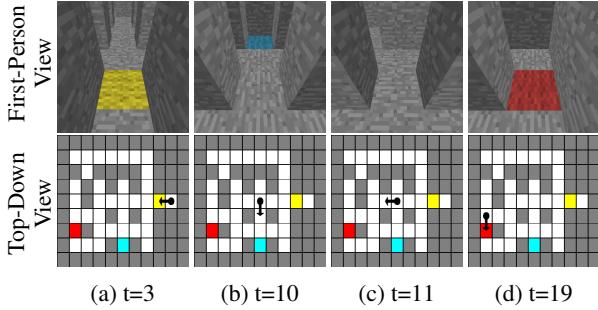


Figure 1. Example task in Minecraft. In this task, the agent should visit the red block if the indicator (next to the start location) is yellow. Otherwise, if the indicator is green, it should visit the blue block. The top row shows the agent’s first-person observation. The bottom row visualizes the map and the agent’s location; this is not available to the agent. (a) The agent observes the yellow indicator. (b) The agent looks left and sees the blue block, (c) but it decides to keep going straight having previously seen the yellow indicator. (d) Finally, it visits the red block and receives a positive reward.

Recently, researchers have explored problems that require faculties associated with higher-level cognition (e.g., inferring simple general purpose algorithms: Graves et al., 2014, and, Q&A: Weston et al., 2015). Most of these advances, however, are restricted to the supervised learning setting, which provides clear error signals. In this paper, we are interested in extending this success to similarly cognition-inspired RL tasks. Specifically, this paper introduces a set of tasks in *Minecraft*¹, a flexible 3D world in which an agent can collect resources, build structures, and survive attacks from enemies. Our RL tasks (one example is illustrated in Figure 1) not only have the usual RL challenges of partial observability, high-dimensional (visual) perception, and delayed reward, but also require an agent to develop movement policies by learning how to use its active perception to observe useful information and collect reward. In addition, our RL tasks require an agent to learn to use any memory it possesses including its interaction with active perception which feeds observations into

¹<https://minecraft.net/>

memory. We note that for simplicity we hereafter refer to these cognition-inspired tasks as cognitive tasks but acknowledge that they form at best a very limited exploration of the range of cognitive faculties in humans.

In this work, we aim to not only systematically evaluate the performance of different neural network architectures on our tasks, but also examine how well such architectures generalize to unseen or larger topologies (Minecraft maps). The empirical results show that existing DRL architectures (Mnih et al., 2015; Hausknecht & Stone, 2015) perform worse on unseen or larger maps compared to training sets of maps, even though they perform reasonably well on the training maps. Motivated by the lack of generalization of existing architectures on our tasks, we also propose new memory-based DRL architectures. Our proposed architectures store recent observations into their memory and retrieve relevant memory based on the temporal context, whereas memory retrieval in existing architectures used in RL problems is not conditioned on the context. In summary, we show that our architectures outperform existing ones on most of the tasks as well as generalize better to unseen maps by exploiting their new memory mechanisms.

2. Related Work

Neural Networks with External Memory. Graves et al. (2014) introduced a Neural Turing Machine (NTM), a differentiable external memory architecture, and showed that it can learn algorithms such as copy and reverse. Zaremba & Sutskever (2015) proposed RL-NTM that has a non-differentiable memory to scale up the addressing mechanism of NTM and applied policy gradient to train the architecture. Joulin & Mikolov (2015) implemented a stack using neural networks and demonstrated that it can infer several algorithmic patterns. Sukhbaatar et al. (2015b) proposed a Memory Network (MemNN) for Q&A and language modeling tasks, which stores all inputs and retrieves relevant memory blocks depending on the question.

Deep Reinforcement Learning. Neural networks have been used to learn features for RL tasks for a few decades (e.g., Tesauro, 1995 and Lange & Riedmiller, 2010). Recently, Mnih et al. (2015) proposed a Deep Q-Network (DQN) for training deep convolutional neural networks (CNNs) through Q-Learning in an end-to-end fashion; this achieved state-of-the-art performance on Atari games. Guo et al. (2014) used slow Monte-Carlo Tree Search (MCTS) (Kocsis & Szepesvári, 2006) to generate a relatively small amount of data to train fast-playing convolutional networks in Atari games. Schulman et al. (2015), Levine et al. (2016), and Lillicrap et al. (2016) have successfully trained deep neural networks to directly learn policies and applied their architectures to robotics problems. In addition, there are deep RL approaches to tasks other than Atari such as learning algorithms (Zaremba

et al., 2016) and text-based games (Sukhbaatar et al., 2015a; Narasimhan et al., 2015). There have also been a few attempts to learn state-transition models using deep learning to improve exploration in RL (Oh et al., 2015; Stadie et al., 2015). Most recently, Mnih et al. (2016) proposed asynchronous DQN and showed that it can learn to explore a 3D environment similar to Minecraft. Unlike their work, we focus on a systematic evaluation of the ability to deal with partial observability, active perception, and external memory in different neural network architectures as well as generalization across size and maps.

Model-free Deep RL for POMDPs. Building a model-free agent in partially observable Markov decision processes (POMDPs) is a challenging problem because the agent needs to learn how to summarize history for action-selection. To deal with such a challenge, Bakker et al. (2003) used a Long Short-Term Memory (LSTM) network (Hochreiter & Schmidhuber, 1997) in an offline policy learning framework to show that a robot controlled by an LSTM network can solve *T-Mazes* where the robot should go to the correct destination depending on the traffic signal at the beginning of the maze. Wierstra et al. (2010) proposed a *Recurrent Policy Gradient* method and showed that an LSTM network trained using this method outperforms other methods in several tasks including T-Mazes. More recently, Zhang et al. (2016) introduced continuous memory states to augment the state and action space and showed it can memorize salient information through *Guided Policy Search* (Levine & Koltun, 2013). Hausknecht & Stone (2015) proposed Deep Recurrent Q-Network (DRQN) which consists of an LSTM on top of a CNN based on the DQN framework and demonstrated improved handling of partial observability in Atari games.

Departure from Related Work. The architectures we introduce use memory mechanisms similar to MemNN, but our architectures have a layer that constructs a query for memory retrieval based on temporal context. Our architectures are also similar to NTM in that a recurrent controller interacts with an external memory, but ours have a simpler writing and addressing mechanism which makes them easier to train. Most importantly, our architectures are used in an RL setting and must learn from a delayed reward signal, whereas most previous work in exploring architectures with memory is in the supervised learning setting with its much more direct and undelayed error signals. We describe details of our architectures in Section 4.

The tasks we introduce are inspired by the T-maze experiments (Bakker et al., 2003) as well as Maze-Base (Sukhbaatar et al., 2015a), which has natural language descriptions of mazes available to the agent. Unlike these previous tasks, our mazes have high-dimensional visual observations with deep partial observability due to the nature of the 3D worlds. In addition, the agent has to learn how

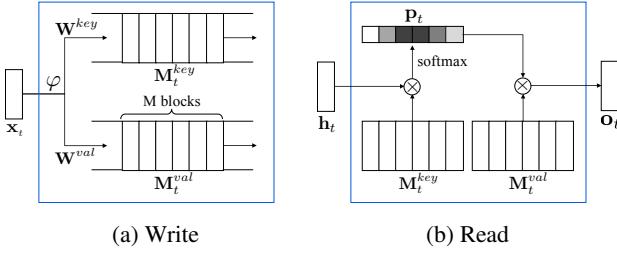


Figure 2. Illustration of memory operations.

best to control its active perception system to collect useful information at the right time in our tasks; this is not necessary in previous work.

3. Background: Deep Q-Learning

Denote the state, immediate reward, and action at time t as s_t, r_t, a_t respectively. In the DQN framework, every transition $T_t = (s_t, s_{t+1}, a_t, r_t)$ is stored in a *replay memory*. For (each) iteration i , the deep neural network (with parameters θ) is trained to approximate the action-value function from transitions $\{(s, s', a, r)\}$ by minimizing the loss functions $L_i(\theta_i)$ as follows:

$$\begin{aligned}\mathcal{L}_i(\theta) &= \mathbb{E}_{s,a \sim \pi_\theta} [(y_i - Q(s, a; \theta))^2] \\ \nabla_\theta \mathcal{L}_i(\theta) &= \mathbb{E}_{s,a \sim \pi_\theta} [(y_i - Q(s, a; \theta)) \nabla_\theta Q(s, a; \theta)]\end{aligned}$$

where $y_i = \mathbb{E}_{s' \sim \pi_\theta} [r + \gamma \max_{a'} Q(s', a'; \theta')]$ is the target Q-value estimated by a *target Q-network* (θ'). In practice, the expectation terms are approximated by sampling a mini-batch of transitions from the replay memory. The parameter of target Q-network (θ') is synchronized with the learned network (θ) after a fixed number of iterations.

4. Architectures

The importance of retrieving a prior observation from memory depends on the current context. For example, in the maze of Figure 1 where the color of the indicator block determines the desired target color, the indicator information is important only when the agent is seeing a potential target and has to decide whether to approach it or find a different target. Motivated by the lack of “context-dependent memory retrieval” in existing DRL architectures, we present three new memory-based architectures in this section.

Our proposed architectures (Figure 3c-e) consist of convolutional networks for extracting high-level features from images (§4.1), a memory that retains a recent history of observations (§4.2), and a context vector used both for memory retrieval and (in part for) action-value estimation (§4.3). Depending on how the context vector is constructed, we obtain three new architectures: Memory Q-Network (MQN), Recurrent Memory Q-Network (RMQN), and Feedback Recurrent Memory Q-Network (FRMQN).

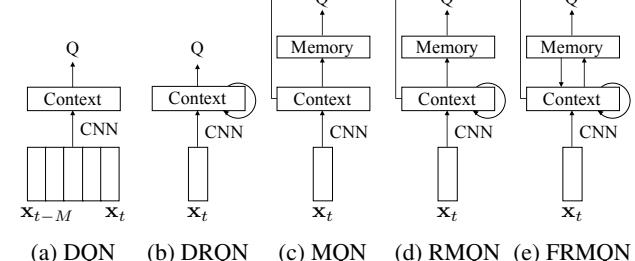


Figure 3. Illustration of different architectures

4.1. Encoding

For each time-step, a raw observation (pixels) is encoded to a fixed-length vector as follows:

$$\mathbf{e}_t = \varphi^{enc}(\mathbf{x}_t) \quad (1)$$

where $\mathbf{x}_t \in \mathbb{R}^{c \times h \times w}$ is $h \times w$ image with c channels, and $\mathbf{e}_t \in \mathbb{R}^e$ is the encoded feature at time t . In this work, we use a CNN to encode the observation.

4.2. Memory

The memory operations in the proposed architectures are similar to those proposed in MemNN.

Write. The encoded features of last M observations are linearly transformed and stored into the memory as *key* and *value* memory blocks as illustrated in Figure 2a. More formally, two types of memory blocks are defined as follows:

$$\mathbf{M}_t^{key} = \mathbf{W}^{key} \mathbf{E}_t \quad (2)$$

$$\mathbf{M}_t^{val} = \mathbf{W}^{val} \mathbf{E}_t \quad (3)$$

where $\mathbf{M}_t^{key}, \mathbf{M}_t^{val} \in \mathbb{R}^{m \times M}$ are memory blocks with m -dimensional embeddings, and $\mathbf{W}^{key}, \mathbf{W}^{val} \in \mathbb{R}^{m \times e}$ are parameters of the linear transformations for keys and values respectively. $\mathbf{E}_t = [\mathbf{e}_{t-1}, \mathbf{e}_{t-2}, \dots, \mathbf{e}_{t-M}] \in \mathbb{R}^{e \times M}$ is the concatenation of features of the last M observations.

Read. The reading mechanism of the memory is based on soft attention (Graves, 2013; Bahdanau et al., 2015) as illustrated in Figure 2b. Given a context vector $\mathbf{h}_t \in \mathbb{R}^m$ (§4.3), the memory module draws soft attention over memory locations (and implicitly time) by computing the inner-product between the context and all key memory blocks as follows:

$$p_{t,i} = \frac{\exp(\mathbf{h}_t^\top \mathbf{M}_t^{key}[i])}{\sum_{j=1}^M \exp(\mathbf{h}_t^\top \mathbf{M}_t^{key}[j])} \quad (4)$$

where $p_{t,i} \in \mathbb{R}$ is an attention weight for i -th memory block ($t-i$ time-step). The output of the read operation is the linear sum of the value memory blocks based on the attention weights as follows:

$$\mathbf{o}_t = \mathbf{M}_t^{val} \mathbf{p}_t \quad (5)$$

where $\mathbf{o}_t \in \mathbb{R}^m$ and $\mathbf{p}_t \in \mathbb{R}^M$ are the retrieved memory and the attention weights respectively.

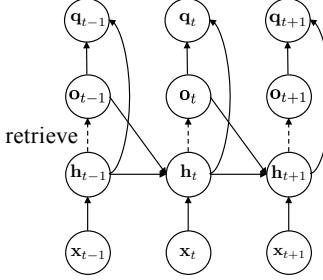


Figure 4. Unrolled illustration of FRMQN.

4.3. Context

To retrieve useful information from memory, the context vector should capture relevant spatio-temporal information from the observations. To this end, we present three different architectures for constructing the context vector:

$$\text{MQN: } \mathbf{h}_t = \mathbf{W}^c \mathbf{e}_t \quad (6)$$

$$\text{RMQN: } [\mathbf{h}_t, \mathbf{c}_t] = \text{LSTM}(\mathbf{e}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1}) \quad (7)$$

$$\text{FRMQN: } [\mathbf{h}_t, \mathbf{c}_t] = \text{LSTM}([\mathbf{e}_t, \mathbf{o}_{t-1}], \mathbf{h}_{t-1}, \mathbf{c}_{t-1}) \quad (8)$$

where $\mathbf{h}_t, \mathbf{c}_t \in \mathbb{R}^m$ are a context vector and a memory cell of LSTM respectively, and $[\mathbf{e}_t, \mathbf{o}_{t-1}]$ denotes concatenation of the two vectors as input for LSTM. **MQN** is a feed-forward architecture that constructs the context based on only the current observation, which is very similar to MemNN except that the current input is used for memory retrieval in the temporal context of an RL problem. **RMQN** is a recurrent architecture that captures spatio-temporal information from the history of observations using LSTM. This architecture allows for retaining temporal information through LSTM as well as external memory. Finally, **FRMQN** has a feedback connection from the retrieved memory to the context vector as illustrated in Figure 4. This allows the FRMQN architecture to refine its context based on the previously retrieved memory so that it can do more complex reasoning as time goes on. Note that feedback connections are analogous to the idea of *multiple hops* in MemNN in the sense that the architecture retrieves memory blocks multiple times based on the previously retrieved memory. However, FRMQN retrieves memory blocks through time, while MemNN does not.

Finally, the architectures estimate action-values by incorporating the retrieved memory and the context vector:

$$\mathbf{q}_t = \varphi^q(\mathbf{h}_t, \mathbf{o}_t) \quad (9)$$

where $\mathbf{q}_t \in \mathbb{R}^a$ is the estimated action-value, and φ^q is a multi-layer perceptron (MLP) taking two inputs. In the results we report here, we used an MLP with one hidden layer as follows: $\mathbf{g}_t = f(\mathbf{W}^h \mathbf{h}_t + \mathbf{o}_t)$, $\mathbf{q}_t = \mathbf{W}^q \mathbf{g}_t$ where f is a rectified linear function (Nair & Hinton, 2010) applied only to half of the hidden units for easy optimization by following Sukhbaatar et al. (2015b).

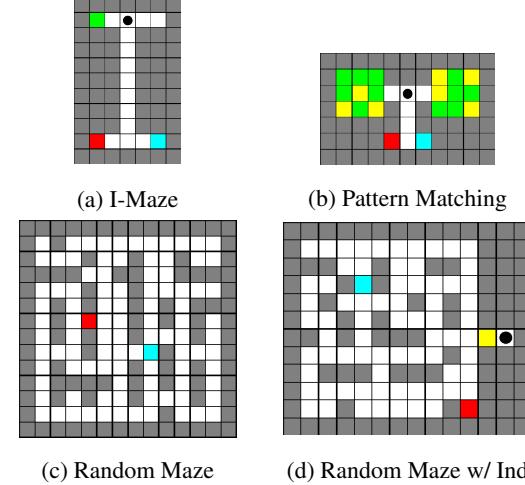


Figure 5. Examples of maps. (a) has an I-structured topology where the location of indicator (yellow/green), goals (red/blue), and spawn locations (black circle) are fixed across episodes. (b) has two goals and two rooms with color patterns. (c) consists of randomly generated walls and two goals. The agent can be spawned anywhere except for goal locations. (d) is similar to (c) except that it has an indicator at the fixed location (yellow/green) and a fixed spawn location.

5. Experiments

The experiments, baselines, and tasks are designed to investigate how useful context-dependent memory retrieval is for generalizing to unseen maps, and when memory feedback connections in FRMQN are helpful. Game play videos can be found in the supplementary material and at the following website: <https://sites.google.com/a/umich.edu/junhyuk-oh/icml2016-minecraft>. Next, we describe aspects that are common to all tasks and our training methodology.

Environment. In all the tasks, episodes terminate either when the agent finishes the task or after 50 steps. An agent receives -0.04 reward at every time step. The agent's initial looking direction is randomly selected among four directions: north, south, east, and west. For tasks where there is randomness (e.g., maps, spawn points), we randomly sampled an instance after every episode.

Actions. The following six actions are available: Look left/right ($\pm 90^\circ$ in yaw), Look up/down ($\pm 45^\circ$ in pitch), and Move forward/backward. Moving actions move the agent one block forward or backward in the direction it is facing. The pitch is limited to $[-45^\circ, 0^\circ]$.

Baselines. We compare our three architectures with two baselines: **DQN** (Mnih et al., 2015) (see Figure 3a) and **DRQN** (Hausknecht & Stone, 2015) (see Figure 3b). DQN is a CNN architecture that takes a fixed number of frames as input. DRQN is a recurrent architecture that has an LSTM layer on top of the CNN. Note that DQN cannot take more

Table 1. Performance on I-Maze. Each entry shows the average success rate with standard error measured from 10 runs. For each run, we measured the average success rate of 10 best-performing parameters based on the performance on unseen set of maps. The success rate is defined as the number of episodes that the agent reaches the correct goal within 100 steps divided by the total number of episodes. ‘Size’ represents the number of blocks of the vertical corridor. ‘✓’ indicates that such sizes of I-Mazes belong to the training set of maps.

SIZE	TRAIN	DQN	DRQN	MQN	RMQN	FRMQN
4		92.1(1.5)	94.8(1.5)	87.2(2.3)	89.2(2.4)	96.9(1.0)
5	✓	99.3(0.5)	98.2(1.1)	96.2(1.0)	98.6(0.5)	99.3(0.7)
6		99.4(0.4)	98.2(1.0)	96.0(1.0)	99.0(0.4)	99.7(0.3)
7	✓	99.6(0.3)	98.8(0.8)	98.0(0.6)	98.8(0.5)	100.0(0.0)
8		99.3(0.4)	98.3(0.8)	98.3(0.5)	98.0(0.8)	100.0(0.0)
9	✓	99.0(0.5)	98.4(0.6)	98.0(0.7)	94.6(1.8)	100.0(0.0)
10		96.5(0.7)	97.4(1.1)	98.2(0.7)	87.5(2.6)	99.6(0.3)
15		50.7(0.9)	83.3(3.2)	96.7(1.3)	89.8(2.4)	97.4(1.1)
20		48.3(1.0)	63.6(3.7)	97.2(0.9)	96.3(1.2)	98.8(0.5)
25		48.1(1.0)	57.6(3.7)	98.2(0.7)	90.3(2.5)	98.4(0.6)
30		48.6(1.0)	60.5(3.6)	97.9(0.9)	87.1(2.4)	98.1(0.6)
35		49.5(1.2)	59.0(3.4)	95.0(1.1)	84.0(3.2)	94.8(1.2)
40		46.6(1.2)	59.2(3.6)	77.2(4.2)	71.3(5.0)	89.0(2.6)

than the number of frames used during training because its first convolution layer takes a fixed number of observations. However, DRQN and our architectures can take arbitrary number of input frames using their recurrent layers. Additionally, our architectures can use an arbitrarily large size of memory during evaluation as well.

Training details. Input frames from Minecraft are captured as 32×32 RGB images. All the architectures use the same 2-layer CNN architecture as described in the supplementary material. In the DQN and DRQN architectures, the last convolutional layer is followed by a fully-connected layer with 256 hidden units. In our architectures, the last convolution layer is given as the encoded feature for memory blocks. In addition, 256 LSTM units are used in DRQN, RMQN, and FRMQN. More details including hyperparameters for Deep Q-Learning are described in the supplementary material. Our implementation is based on Torch7 (Collobert et al., 2011), a public DQN implementation (Mnih et al., 2015), and a Minecraft Forge Mod.²

5.1. I-Maze: Description and Results

Task. Our I-Maze task was inspired by T-Mazes which have been used in animal cognition experiments (Olton, 1979). Maps for this task (see Figure 5a) have an indicator at the top that has equal chance of being yellow or green. If the indicator is yellow, the red block gives +1 reward and the blue block gives -1 reward; if the indicator is green, the red block gives -1 and the blue block gives +1 reward. Thus, the agent should memorize the color of the indicator at the beginning while it is in view and visit the correct goal depending on the indicator-color. We varied the length of the vertical corridor to $l = \{5, 7, 9\}$ during training. The last 12 frames were given as input for all architectures, and

the size of memory for our architectures was 11.

Performance on the training set. We observed two stages of behavior during learning from all the architectures: 1) early in the training the discount factor and time penalty led to the agent to take a chance by visiting any goal, and 2) later in the training the agent goes to the correct goal by learning the correlation between the indicator and the goal. As seen in the learning curves in Figure 6a, our architectures converge more quickly than DQN and DRQN to the correct behavior. In particular, we observed that DRQN takes many more epochs to reach the second stage after the first stage has been reached. This is possibly due to the long time interval between seeing the indicator and the goals. Besides, the indicator block is important only when the agent is at the bottom end of the vertical corridor and needs to decide which way to go (see Figure 5a). In other words, the indicator information does not affect the agent’s decision making along its way to the end of the corridor. This makes it even more difficult for DRQN to retain the indicator information for a long time. On the other hand, our architectures can handle these problems by storing the history of observations into memory and retrieving such information when it is important, based on the context.

Generalization performance. To investigate generalization performance, we evaluated the architectures on maps that have vertical corridor lengths $\{4, 6, 8, 10, 15, 20, 25, 30, 35, 40\}$ that were not present in the training maps. More specifically, testing on $\{6, 8\}$ sizes of maps and the rest of the sizes of maps can evaluate *interpolation* and *extrapolation* performance, respectively (Schaul et al., 2015). Since some unseen maps are larger than the training maps, we used 50 last frames as input during evaluation on the unseen maps for all architectures except for DQN, which can take only 12 frames as discussed in the experimental setup. The size of memory for our architectures is set to 49. The performance on the unseen set of maps is visualized in Figure 6b. Although the generalization performances of all architectures are highly variable even after training performance converges, it can be seen that FRMQN consistently outperforms the other architectures in terms of average reward. To further investigate the performance for different lengths of the vertical corridor, we measured the performance on each size of map in Table 1. It turns out that all architectures perform well on $\{6, 8\}$ sizes of maps, which indicates that they can interpolate within the training set of maps. However, our architectures extrapolate to larger maps significantly better than the two baselines.

Analysis of memory retrieval. Figure 7a visualizes FRMQN’s memory retrieval on a large I-Maze, where FRMQN sharply retrieves the indicator information only when it reaches the end of the corridor where it then makes a decision of which goal block to visit. This is a reasonable strategy because the indicator information is important only

²<http://files.minecraftforge.net/>

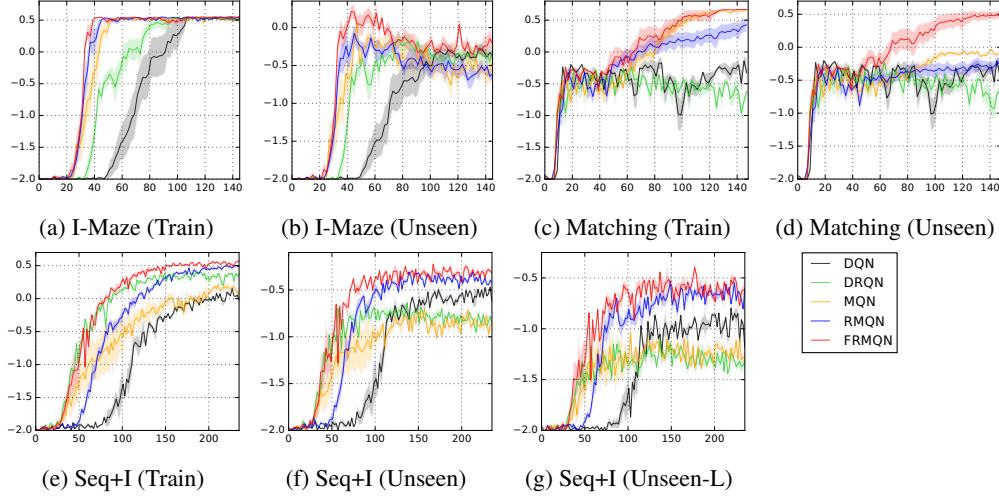


Figure 6. Learning curves for different tasks: (a-b) I-maze (§5.1), (c-d) pattern matching (§5.2), (e-g) random mazes (§5.3). X-axis and y-axis correspond to the number of training epochs (1 epoch = 10K steps) and the average reward. For (b) and (d), ‘Unseen’ represents unseen maps with different sizes and different patterns respectively. For (f) and (g), ‘Unseen’ and ‘Unseen-L’ indicate unseen topologies with the same sizes and larger sizes of maps, respectively. The performance was measured from 4 runs for random mazes and 10 runs for I-Maze and Pattern Matching. For the random mazes, we only show the results on Sequential Goals with Indicator due to space constraints. More plots are provided in the supplementary material.

Table 2. Performance on pattern matching. The entries represent the probability of visiting the correct goal block for each set of maps with standard error. The performance reported is averages over 10 runs and 10 best-performing parameters for each run.

	TRAIN	UNSEEN
DQN	62.9% ($\pm 3.4\%$)	60.1% ($\pm 2.8\%$)
DRQN	49.7% ($\pm 0.2\%$)	49.2% ($\pm 0.2\%$)
MQN	99.0% ($\pm 0.2\%$)	69.3% ($\pm 1.5\%$)
RMQN	82.5% ($\pm 2.5\%$)	62.3% ($\pm 1.5\%$)
FRMQN	100.0% ($\pm 0.0\%$)	91.8% ($\pm 1.0\%$)

when it is at the end of the vertical corridor. This qualitative result implies that FRMQN learned a general strategy that looks for the indicator, goes to the end of the corridor, and retrieves the indicator information when it decides which goal block to visit. We observed similar policies learned by MQN and RMQN, but the memory attention for the indicator was not as sharp as FRMQN’s attention and so they visit wrong goals in larger I-Mazes more often.

The results on I-Maze shown above suggest that solving a task on a set of maps does not guarantee solving the same task on similar but unseen maps, and such generalization performance highly depends on the feature representation learned by deep neural networks. The extrapolation result shows that context-dependent memory retrieval in our architectures is important for learning a general strategy when the importance of an observational-event depends highly on the temporal context.

5.2. Pattern Matching: Description and Results

Task. As illustrated in Figure 5b, this map consists of two 3×3 rooms. The visual patterns of the two rooms are either identical or different with equal probability. If the two

rooms have the exact same color patterns, the agent should visit the blue block. If the rooms have different color patterns, the agent should visit the red block. The agent receives a +1 reward if it visits the correct block and a -1 reward if it visits the wrong block. This pattern matching task requires more complex reasoning (comparing two visual patterns given at different time steps) than the I-Maze task above. We generated 500 training and 500 unseen maps in such a way that there is little overlap between the two sets of visual patterns. Details of the map generation process are described in the supplementary material. The last 10 frames were given as input for all architectures, and the size of memory was set to 9.

Performance on the training set. The results plotted in Figure 6c and Table 2 show that MQN and FRMQN successfully learned to go to the correct goal block for all runs in the training maps. We observed that DRQN always learned a sub-optimal policy that goes to any goal regardless of the visual patterns of the two rooms. Another observation is the training performances of DQN and RMQN are a bit unstable; they often learned the same sub-optimal policy, whereas MQN and FRMQN consistently learned to go to the correct goal across different runs. We hypothesize that it is not trivial for a neural network to compare two visual patterns observed in different time-steps unless the network can model high-order interactions between two specific observations for visual matching, which might be the reason why DQN and DRQN fail more often. Context-dependent memory retrieval mechanism in our architectures can alleviate this problem by retrieving two visual patterns corresponding to the observations of the two rooms before decision making.

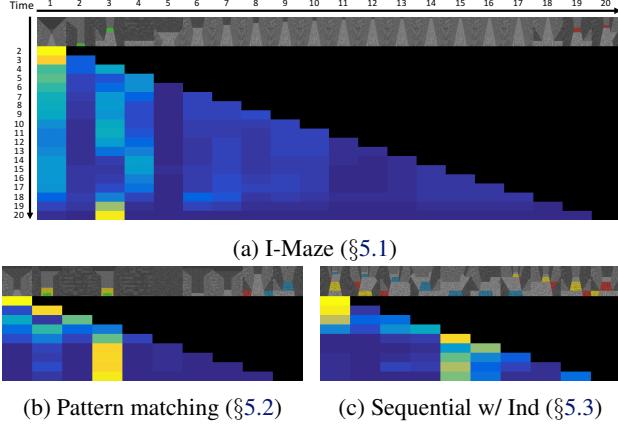


Figure 7. Visualization of FRMQN’s memory retrieval. Each figure shows a trajectory of FRMQN at the top row, and the following rows visualize attention weights over time. (a) The agent looks at the indicator, goes to the end of the corridor, and retrieves the indicator frame before visiting the goal block. (b) The agent looks at both rooms at the beginning and gradually switches attention weights from one room to another room as it approaches the goal blocks. (c) The agent pays attention to the indicator (yellow) and the first goal block (blue).

Generalization performance. Table 2 and Figure 6d show that FRMQN achieves the highest success rate on the unseen set of maps. Interestingly, MQN fails to generalize to unseen visual patterns. We observed that MQN pays attention to the two visual patterns before choosing one of the goals through its memory retrieval. However, since the retrieved memory is just a convex combination of two visual patterns, it is hard for MQN to compare the similarity between them. Thus, we believe that MQN simply overfits to the training maps by memorizing the weighted sum of pairs of visual patterns in the training set of maps. On the other hand, FRMQN can utilize retrieved memory as well as its recurrent connections to compare visual patterns over time.

Analysis of memory retrieval. An example of FRMQN’s memory retrieval is visualized in Figure 7b. FRMQN pays attention to both rooms, gradually moving weight from one to the other as time progresses, which means that the context vector is repeatedly refined based on the encoded features of the room retrieved through its feedback connections. Given this visualization and its good generalization performance, we hypothesize that FRMQN utilizes its feedback connection to compare the two visual features over time rather than comparing them at a single time-step. This result supports our view that feedback connections can play an important role in tasks where more complex reasoning is required with retrieved memories.

5.3. Random Mazes: Description and Results

Task. A random maze task consists of randomly generated walls and goal locations as shown in Figure 5c and 5d. We present 4 classes of tasks using random mazes.

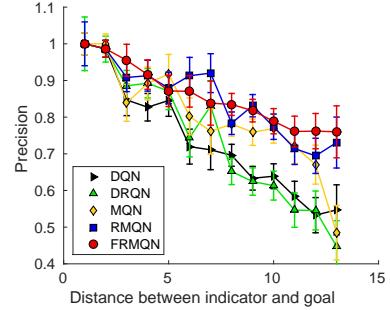


Figure 8. Precision vs. distance. X-axis represents the distance between indicator and goal in Single Goal with Indicator task. Y-axis represents the number of correct goal visits divided by the total number of goal visits.

- **Single Goal:** The task is to visit the blue block which gives +1 reward while avoiding the red block that gives -1 reward.
- **Sequential Goals:** The task is to visit the red block first and then the blue block later which gives +0.5 and +1 reward respectively. If an agent visits the colored blocks in the reverse order, it receives -0.5 and -1 reward respectively.
- **Single Goal with Indicator:** If the indicator is yellow, the task is to visit the red block. If the indicator is green, the task is to visit the blue block. Visiting the correct block results in +1 reward and visiting the incorrect block results in -1 reward.
- **Sequential Goals with Indicator:** If the indicator is yellow, the task is to visit the blue block first and then the red block. If the indicator is green, the task is to visit the red block first and then the blue block. Visiting the blocks in the correct order results in +0.5 for the first block and +1 reward for the second block. Visiting the blocks in the reverse order results in -0.5 and -1 reward respectively.

We randomly generated 1000 maps used for training and two types of unseen evaluation sets of maps: 1000 maps of the same sizes present in the training maps and 1000 larger maps. The last 10 frames were given as input for all architectures, and the size of memory was set to 9.

Performance on the training set. In this task, the agent not only needs to remember important information while traversing the maps (e.g., an indicator) but it also has to search for the goals as different maps have different obstacle and goal locations. Table 3 shows that RMQN and FRMQN achieve higher asymptotic performances than the other architectures on the training set of maps.

Generalization performance. For the larger-sized unseen maps, we terminated episodes after 100 steps rather than 50 steps and used a time penalty of -0.02 considering their size. During evaluation, we used 10 frames as input for DQN and DRQN and 30 frames for MQN, RMQN, and

Table 3. Performance on random maze. The ‘Size’ column lists the size of each set of maps. The entries in the ‘Reward’, ‘Success’, and ‘Fail’ columns are average rewards, success rates, and failure rates measured from 4 runs. We picked the 10 best parameters based on performance on unseen maps for each run and evaluated them on 1000 episodes. ‘Success’ represents the number of correctly completed episodes divided by the total number of episodes, and ‘Fail’ represents the number of incorrectly completed episodes divided by the total number of episodes (e.g., visiting goals in reverse order in sequential goal tasks). The standard errors are lower than 0.03, 1.5%, 1.0% for all average rewards, success rates, and failure rates respectively.

TASK	TYPE	SIZE	DQN			DRQN			MQN			RMQN			FRMQN		
			REWARD	SUCCESS	FAIL	REWARD	SUCCESS	FAIL	REWARD	SUCCESS	FAIL	REWARD	SUCCESS	FAIL	REWARD	SUCCESS	FAIL
SINGLE	TRAIN	4-8	0.31	90.4%	0.6%	0.45	94.5%	0.1%	0.01	78.8%	0.4%	0.49	95.7%	0.1%	0.46	94.6%	0.3%
	UNSEEN	4-8	0.22	87.3%	0.7%	0.23	86.6%	0.2%	0.02	79.4%	0.3%	0.30	89.4%	0.3%	0.26	88.0%	0.5%
	UNSEEN-L	9-14	-0.28	70.0%	0.3%	-0.40	63.0%	0.1%	-0.63	54.3%	0.4%	-0.28	69.3%	0.1%	-0.28	69.0%	0.1%
SEQ	TRAIN	5-7	-0.60	47.6%	0.8%	-0.08	66.0%	0.6%	-0.48	52.1%	0.1%	0.21	77.0%	0.2%	0.22	77.6%	0.2%
	UNSEEN	5-7	-0.66	45.0%	1.0%	-0.54	48.5%	0.9%	-0.59	48.4%	0.1%	-0.13	64.3%	0.1%	-0.18	63.1%	0.3%
	UNSEEN-L	8-10	-0.82	36.6%	1.4%	-0.89	32.6%	1.0%	-0.77	38.9%	0.6%	-0.43	49.6%	1.1%	-0.42	50.8%	1.0%
SINGLE+I	TRAIN	5-7	-0.04	79.3%	6.3%	0.23	87.9%	1.2%	0.11	83.9%	0.7%	0.34	91.7%	0.8%	0.24	88.0%	1.4%
	UNSEEN	5-7	-0.41	64.8%	16.1%	-0.46	61.0%	13.4%	-0.46	64.2%	7.8%	-0.27	70.0%	10.2%	-0.23	71.8%	8.2%
	UNSEEN-L	8-10	-0.74	49.4%	31.6%	-0.98	38.5%	28.3%	-0.66	55.5%	17.1%	-0.39	63.4%	20.4%	-0.43	63.4%	17.2%
SEQ+I	TRAIN	4-6	-0.13	68.0%	7.0%	0.25	78.5%	1.1%	-0.07	67.7%	2.3%	0.37	83.7%	1.0%	0.48	87.4%	0.9%
	UNSEEN	4-6	-0.58	54.5%	14.5%	-0.65	48.8%	9.7%	-0.71	47.3%	7.2%	-0.32	62.4%	7.2%	-0.28	63.8%	7.5%
	UNSEEN-L	7-9	-0.95	39.1%	17.8%	-1.14	30.2%	13.1%	-1.04	34.4%	9.9%	-0.60	49.5%	12.5%	-0.54	51.5%	12.9%

FRMQN; these choices gave the best results for each architecture.

The results in Table 3 show that, as expected, the performance of all the architectures worsen in unseen maps. From the learning curves (see Figure 6e-g), we observed that generalization performance on unseen maps does not improve after some epochs, even though training performance is improving. This implies that improving policies on a fixed set of maps does not necessarily guarantee better performance on new environments. However, RMQN and FRMQN generalize better than the other architectures in most of the tasks. In particular, compared to the other architectures, DRQN’s performance is significantly degraded on unseen maps. In addition, while DQN shows good generalization performance on the Single Goal task which primarily requires search, on the other tasks it tends to go to any goal regardless of important information (e.g., color of indicator). This can be seen through the higher failure rate (the number of incorrectly completed episodes divided by the total number of episodes) of DQN on indicator tasks in Table 3.

To investigate how well the architectures handle partial observability, we measured precision (proportion of correct goal visits to all goal visits) versus the distance between goal and indicator in Single Goal with Indicator task, which is visualized in Figure 8. Notably, the gap between our architectures (RMQN and FRMQN) and the other architectures becomes larger as the distance increases. This result implies that our architectures are better at handling partial observability than the other architectures, because large distance between indicator and goal is more likely to introduce deeper partial observability (i.e., long-term dependency).

Compared to MQN, the RMQN and FRMQN architectures achieve better generalization performance which suggests that the recurrent connections in the latter two architectures

are a crucial component for handling random topologies. In addition, FRMQN and RMQN achieve similar performances, which implies that the feedback connection may not be always helpful in these tasks. We note that given a retrieved memory (e.g., indicator), the reasoning required for these tasks is simpler than the reasoning required for Pattern Matching task.

Analysis of memory retrieval. An example of memory retrieval in FRMQN is visualized in Figure 7c. It retrieves memory that contains important information (e.g., indicator) before it visits a goal block. The memory retrieval strategy is reasonable and is an evidence that the proposed architectures make it easier to generalize to large-scale environments by better handling partial observability.

6. Discussion

In this paper, we introduced three classes of cognition-inspired tasks in Minecraft and compared the performance of two existing architectures with three architectures that we proposed here. We emphasize that unlike most evaluations of RL algorithms, we trained and evaluated architectures on disjoint sets of maps so as to specifically consider the applicability of learned value functions to unseen (interpolation and extrapolation) maps.

In summary, our main empirical result is that context-dependent memory retrieval, particularly with a feedback connection from the retrieved memory, can more effectively solve our set of tasks that require control of active perception and external physical movement actions. Our architectures, particularly FRMQN, also show superior ability relative to the baseline architectures when learning value functions whose behavior generalizes better from training to unseen environments. In future work, we intend to take advantage of the flexibility of the Minecraft domain to construct even more challenging cognitive tasks to further evaluate our architectures.

Acknowledgement

This work was supported by NSF grant IIS-1526059. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsor.

References

- Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.
- Bakker, Bram, Zhumatiy, Viktor, Gruener, Gabriel, and Schmidhuber, Jürgen. A robot that reinforcement-learns to identify and memorize important previous observations. In *Intelligent Robots and Systems*, 2003.
- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 06 2013.
- Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clément. Torch7: A matlab-like environment for machine learning. In *BigLearn, Advances in the Neural Information Processing System Workshop*, 2011.
- Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014.
- Graves, Alex. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.
- Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in the Neural Information Processing System*, 2014.
- Hausknecht, Matthew and Stone, Peter. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposium on Sequential Decision Making for Intelligent Agents*, 2015.
- Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- Joulin, Armand and Mikolov, Tomas. Inferring algorithmic patterns with stack-augmented recurrent nets. In *Advances in the Neural Information Processing System*, 2015.
- Kocsis, Levente and Szepesvári, Csaba. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, 2006.
- Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in the Neural Information Processing System*, 2012.
- Lange, Sascha and Riedmiller, Martin. Deep auto-encoder neural networks in reinforcement learning. In *International Joint Conference on Neural Networks*, 2010.
- LeCun, Yann, Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *Nature*, 521(7553):436–444, 2015.
- Levine, Sergey and Koltun, Vladlen. Guided policy search. In *Proceedings of the International Conference on Machine Learning*, 2013.
- Levine, Sergey, Finn, Chelsea, Darrell, Trevor, and Abbeel, Pieter. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016.
- Lillicrap, Timothy P, Hunt, Jonathan J, Pritzel, Alexander, Heess, Nicolas, Erez, Tom, Tassa, Yuval, Silver, David, and Wierstra, Daan. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.
- Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy P, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2016.
- Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the International Conference on Machine Learning*, 2010.
- Narasimhan, Karthik, Kulkarni, Tejas, and Barzilay, Regina. Language understanding for text-based games using deep reinforcement learning. In *Conference on Empirical Methods on Natural Language Processing*, 2015.

- Oh, Junhyuk, Guo, Xiaoxiao, Lee, Honglak, Lewis, Richard L, and Singh, Satinder. Action-conditional video prediction using deep networks in atari games. In *Advances in the Neural Information Processing System*, 2015.
- Olton, David S. Mazes, maps, and memory. *American Psychologist*, 34(7):583, 1979.
- Schaul, Tom, Horgan, Daniel, Gregor, Karol, and Silver, David. Universal value function approximators. In *Proceedings of the International Conference on Machine Learning*, 2015.
- Schmidhuber, Jürgen. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
- Schulman, John, Levine, Sergey, Moritz, Philipp, Jordan, Michael I, and Abbeel, Pieter. Trust region policy optimization. In *Proceedings of the International Conference on Machine Learning*, 2015.
- Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- Stadie, Bradly C, Levine, Sergey, and Abbeel, Pieter. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.
- Sukhbaatar, Sainbayar, Szlam, Arthur, Synnaeve, Gabriel, Chintala, Soumith, and Fergus, Rob. Mazebase: A sandbox for learning from games. *arXiv preprint arXiv:1511.07401*, 2015a.
- Sukhbaatar, Sainbayar, Weston, Jason, and Fergus, Rob. End-to-end memory networks. In *Advances in the Neural Information Processing System*, 2015b.
- Tesauro, Gerald. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.
- Watkins, Christopher JCH and Dayan, Peter. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.
- Weston, Jason, Chopra, Sumit, and Bordes, Antoine. Memory networks. In *International Conference on Learning Representations*, 2015.
- Wierstra, Daan, Förster, Alexander, Peters, Jan, and Schmidhuber, Jürgen. Recurrent policy gradients. *Logic Journal of IGPL*, 18(5):620–634, 2010.
- Zaremba, Wojciech and Sutskever, Ilya. Reinforcement learning neural turing machines. *arXiv preprint arXiv:1505.00521*, 2015.
- Zaremba, Wojciech, Mikolov, Tomas, Joulin, Armand, and Fergus, Rob. Learning simple algorithms from examples. In *Proceedings of the International Conference on Machine Learning*, 2016.
- Zhang, Marvin, Levine, Sergey, McCarthy, Zoe, Finn, Chelsea, and Abbeel, Pieter. Policy learning with continuous memory states for partially observed robotic control. In *International Conference on Robotics and Automation*, 2016.

A. Implementation Details

A.1. Hyperparameters

For all architectures, the first convolution layer consists of 32, 4×4 , filters with a stride of 2 and a padding of 1. The second convolution layer consists of 64, 4×4 , filters with a stride of 2 and a padding of 1. In Deep Q-Learning, batch size of 32 and discount factor of 0.99 are used. We used a replay memory size of 10^6 for random mazes and 5×10^4 for I-Maze and Pattern Matching tasks. We linearly interpolated ϵ from 1 to 0.1 for the initial 10^6 steps in the ϵ -greedy policy. We chose the best learning rate from $\{0.0001, 0.00025, 0.0005, 0.001\}$ that does not lead to value function explosion depending on the tasks and architectures. The chosen learning rates are shown in Table 4. The parameter is updated after every 4 steps. RMSProp was used with a momentum of 0.95 and a momentum of squared gradients of 0.95. Gradients were clipped at l_2 -norm of 20 to prevent divergence. We used “soft” target Q-network updates with a momentum of 0.999 as suggested by (Lillicrap et al., 2016).

A.2. Map Generation for Pattern Matching

There are a total of 512 possible visual patterns in a 3×3 room with blocks of two colors. We randomly picked 250 patterns and generated two maps for each pattern: one that contains the same pattern in two rooms and another that has a different randomly generated pattern in one of the rooms that is randomly selected. This produces 500 maps, 250 with identical rooms, and 250 with different rooms, which are used for training. For evaluating generalization, we picked another exclusive set of 250 visual patterns, and generated 500 maps by following the same procedure.

Table 4. Learning rates.

TASK	DQN	DRQN	MQN	RMQN	FRMQN
I-MAZE	0.00025	0.0005	0.0005	0.0005	0.0005
MATCHING	0.00025	0.001	0.0005	0.0005	0.0005
SINGLE	0.0001	0.00025	0.0001	0.00025	0.00025
SEQ	0.00025	0.0005	0.00025	0.00025	0.00025
SINGLE+I	0.0001	0.0005	0.00025	0.0005	0.00025
SEQ+I	0.00025	0.001	0.00025	0.00025	0.0005

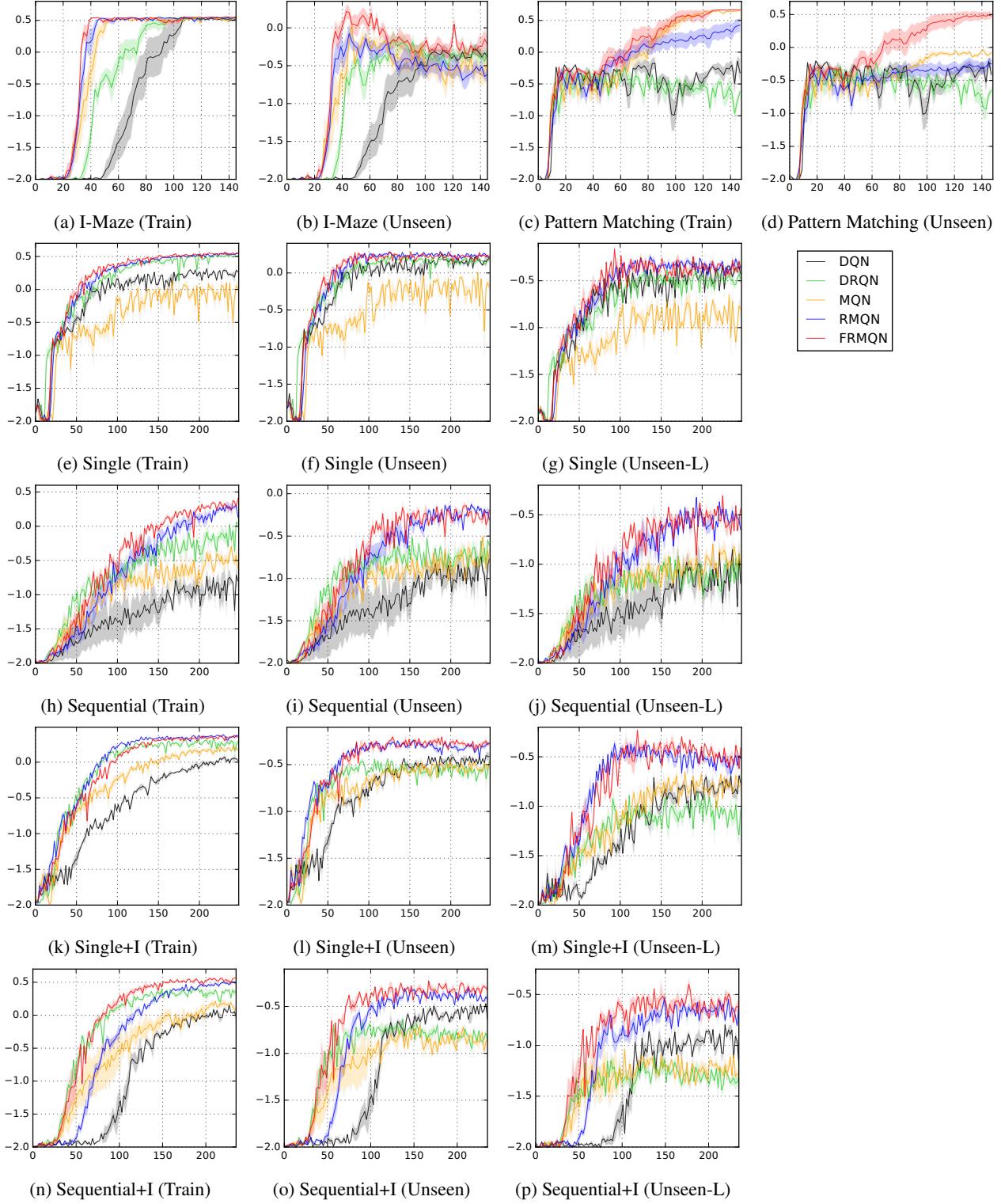


Figure 9. Learning curves. X-axis and y-axis correspond to the number of training epochs (1 epoch = 10K steps) and the average reward respectively. For I-Maze, ‘Unseen’ represents unseen maps with different sizes. For Pattern Matching, ‘Unseen’ represents maps with different visual patterns. For the rest plots, ‘Unseen’ and ‘Unseen-L’ indicate unseen topologies with the same sizes and larger sizes of maps, respectively. The performance was measured from 4 runs for random mazes and 10 runs for I-Maze and Pattern Matching.

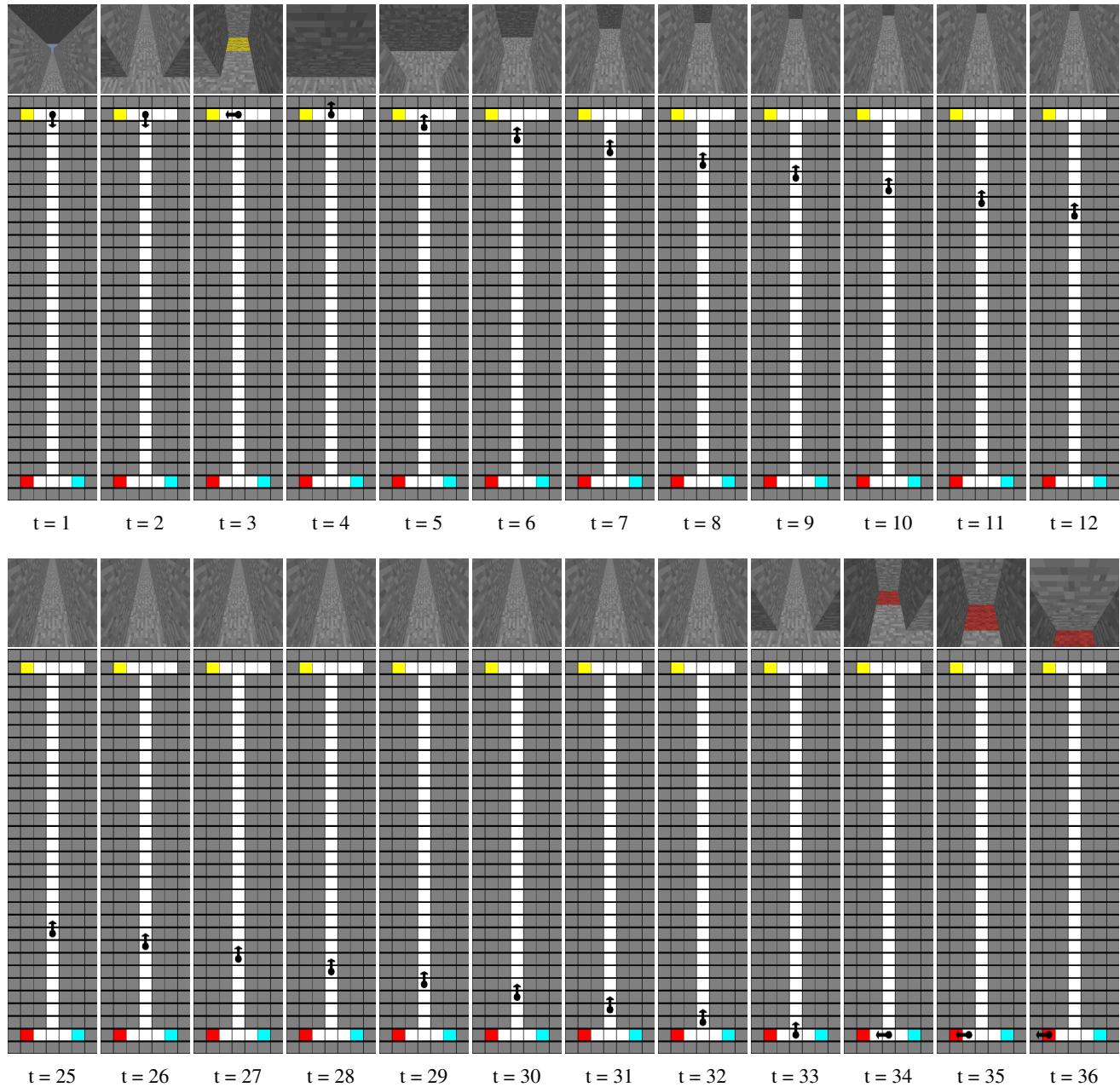


Figure 10. FRMQN’s play on an unseen and larger I-maze. The agent successfully completes the task by visiting the red block given the yellow indicator.

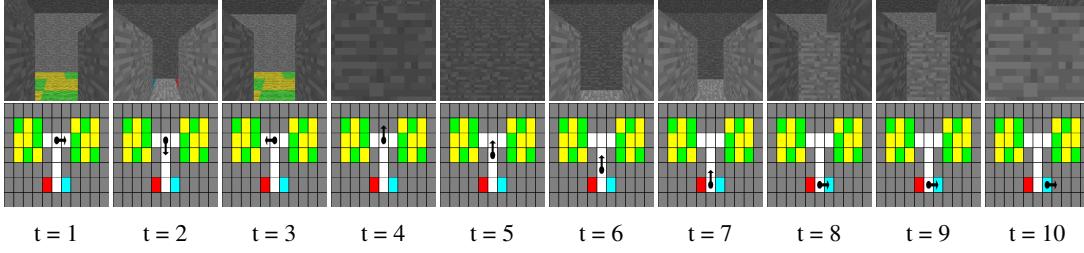


Figure 11. FRMQN’s play on a Pattern Matching task. The agent starts by looking at one room and then turns twice to look at the other room. Upon observing the two rooms, the agent uses backward actions repeatedly to move along the vertical corridor. Finally, once it is at the end of the corridor, it decides to turn and move forward to the blue block as the visual patterns of the rooms were identical. Note that the agent’s performance is near optimal.

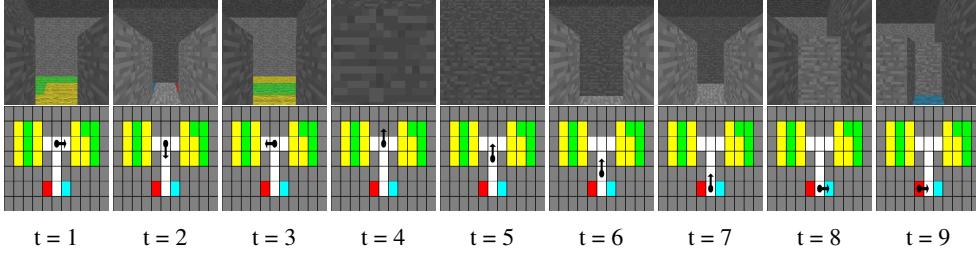


Figure 12. FRMQN’s play on a Pattern Matching task. The agent successfully goes to the red goal, given that the visual patterns of the two rooms were different.

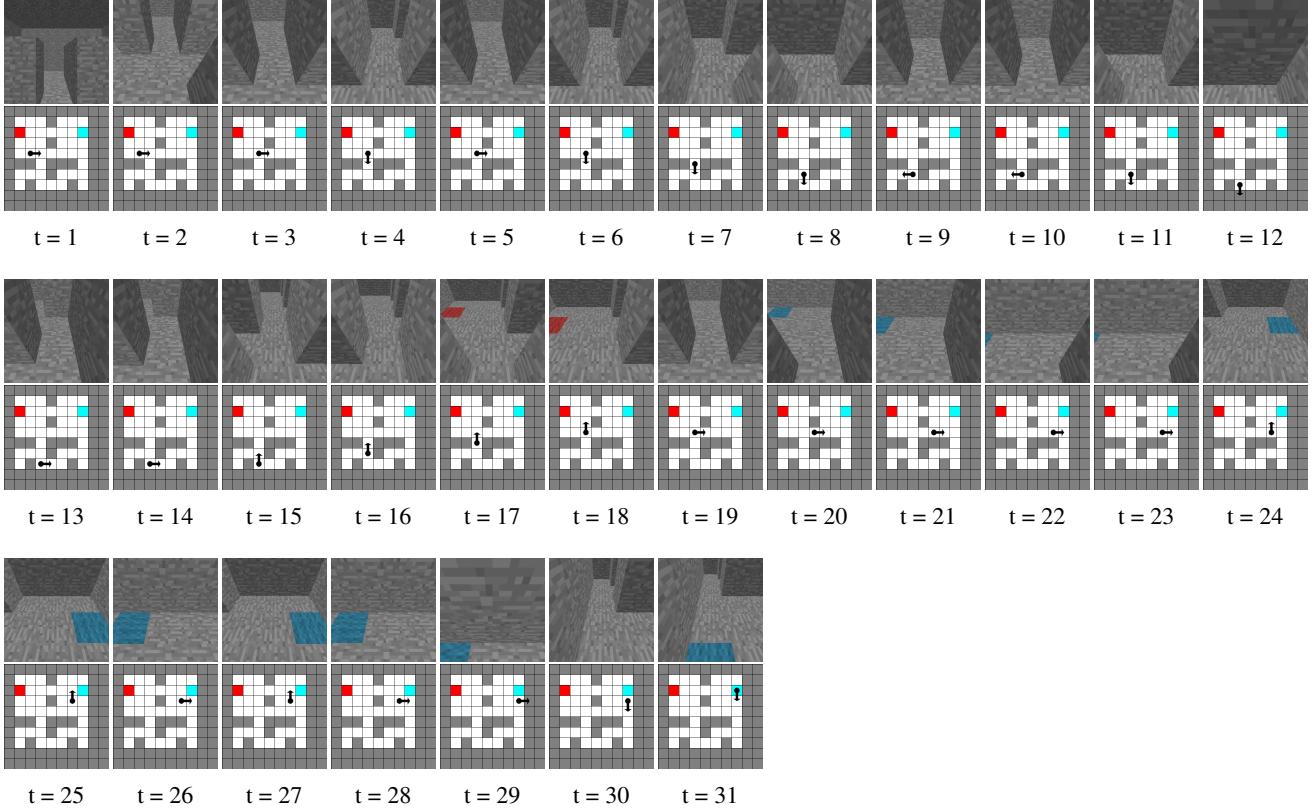


Figure 13. FRMQN’s play on an unseen random maze with Single Goal task. As the case with most of the tasks, the agent starts by looking down quickly to see the important stimuli (e.g., goal blocks and indicators) more clearly. The agent then looks around its vicinity and explores corridors. As soon as the agent sees the blue block, it goes to the block and successfully complete the task.

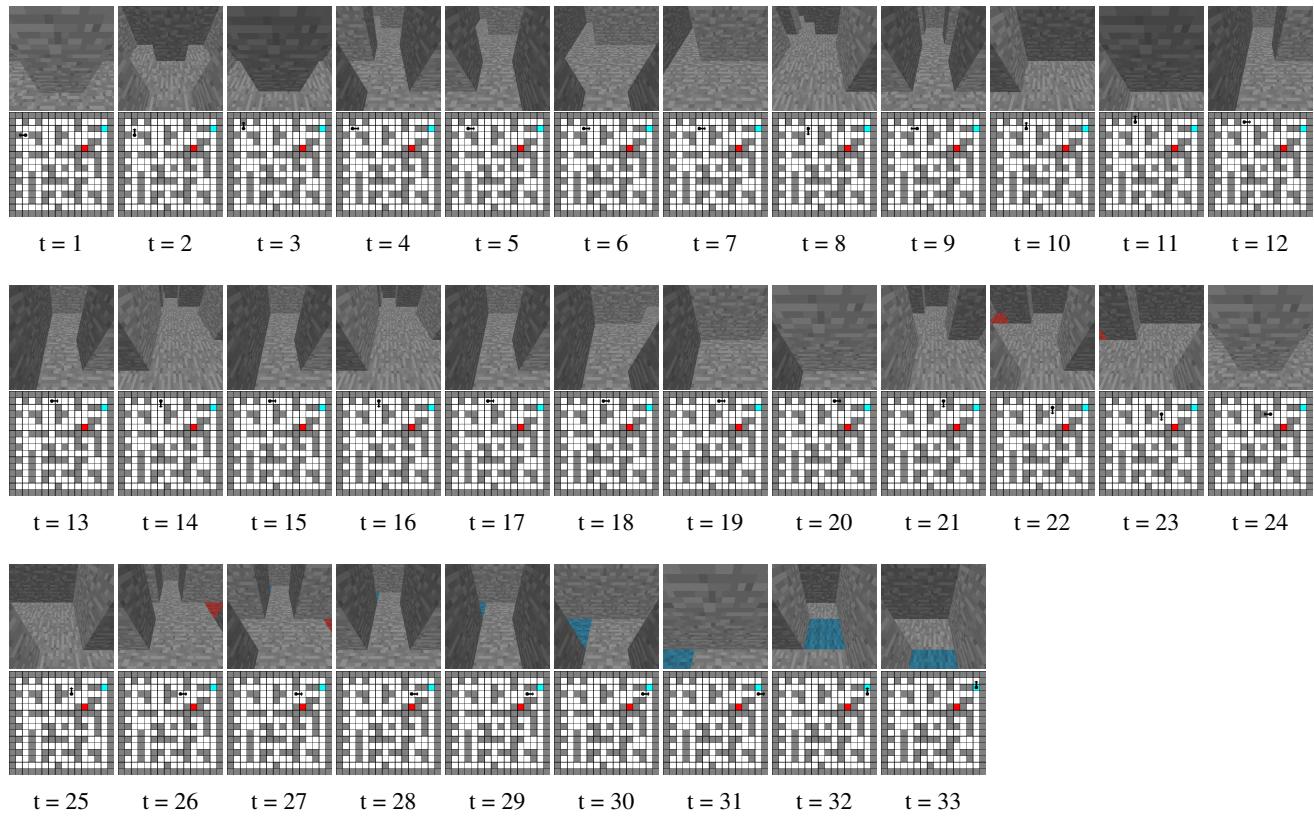


Figure 14. FRMQN’s play in an unseen and larger random maze with Single Goal task. The agent explores the map from the top-left side to top-right side, and successfully finds and visits the blue block.



Figure 15. FRMQN’s play in an unseen and larger random maze with Single Goal task. Even though the agent explores the entire map in a reasonable way, it fails to find the blue block within 100 steps. This can occur occasionally, especially when the map is quite large and intrinsically complex (e.g., contains many walls giving rise to deep partial observability).

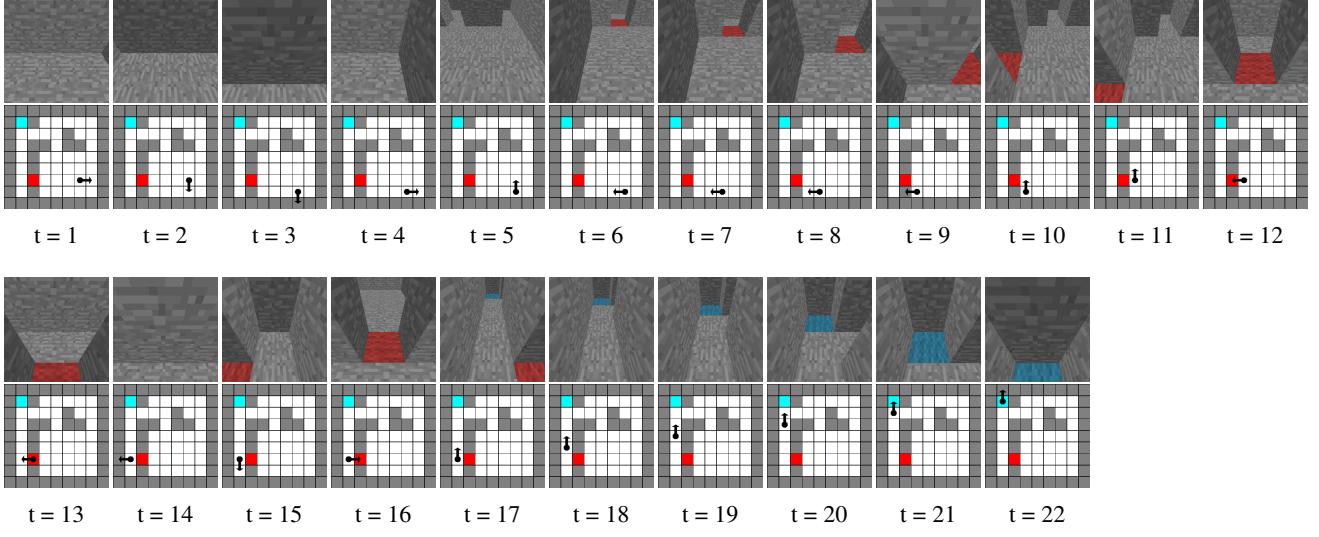


Figure 16. FRMQN’s play on a random maze with Sequential Goals task. As the case with most of the other tasks, the agent begins by looking down. It then looks around for the red block ($t=1-6$). Upon finding the red block, it visits it ($t=13$). The agents then looks for the blue block ($t=14-18$), successfully finds it, and hence complete the sequence and task ($t=22$). Notably the agent does not keep searching for the red block after visiting the red block. This is where memory can be crucial.

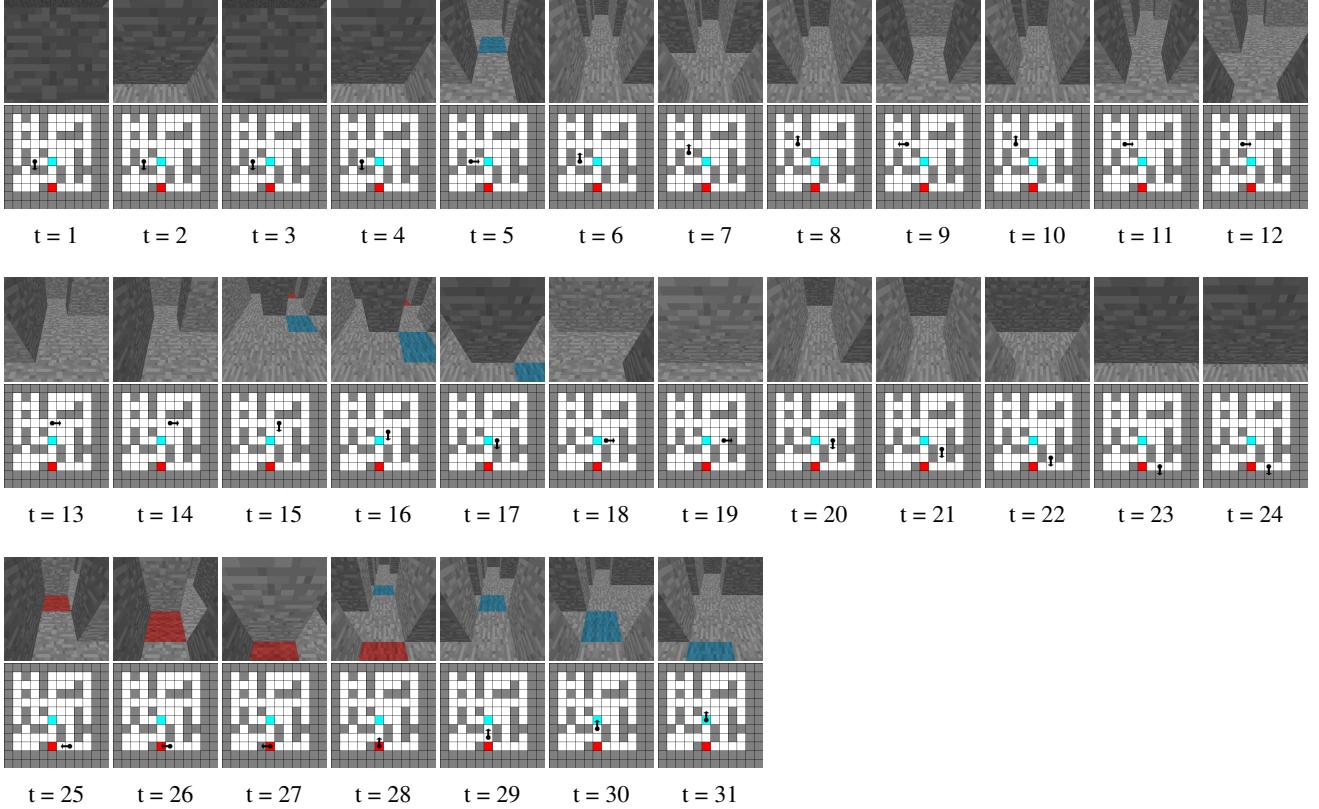


Figure 17. FRMQN’s play on an unseen and larger random maze with Sequential Goals task. With the context of the visual observations containing the blue block ($t=5, 15, 16, 17$), the agent avoids the blue block and keeps searching for the red block based on its memory because it has not visited the red block. After finding and visiting the red block ($t=28$), it directly goes to the blue block ($t=31$), completing the sequence in the correct order, and hence the task.

Control of Memory, Active Perception, and Action in Minecraft

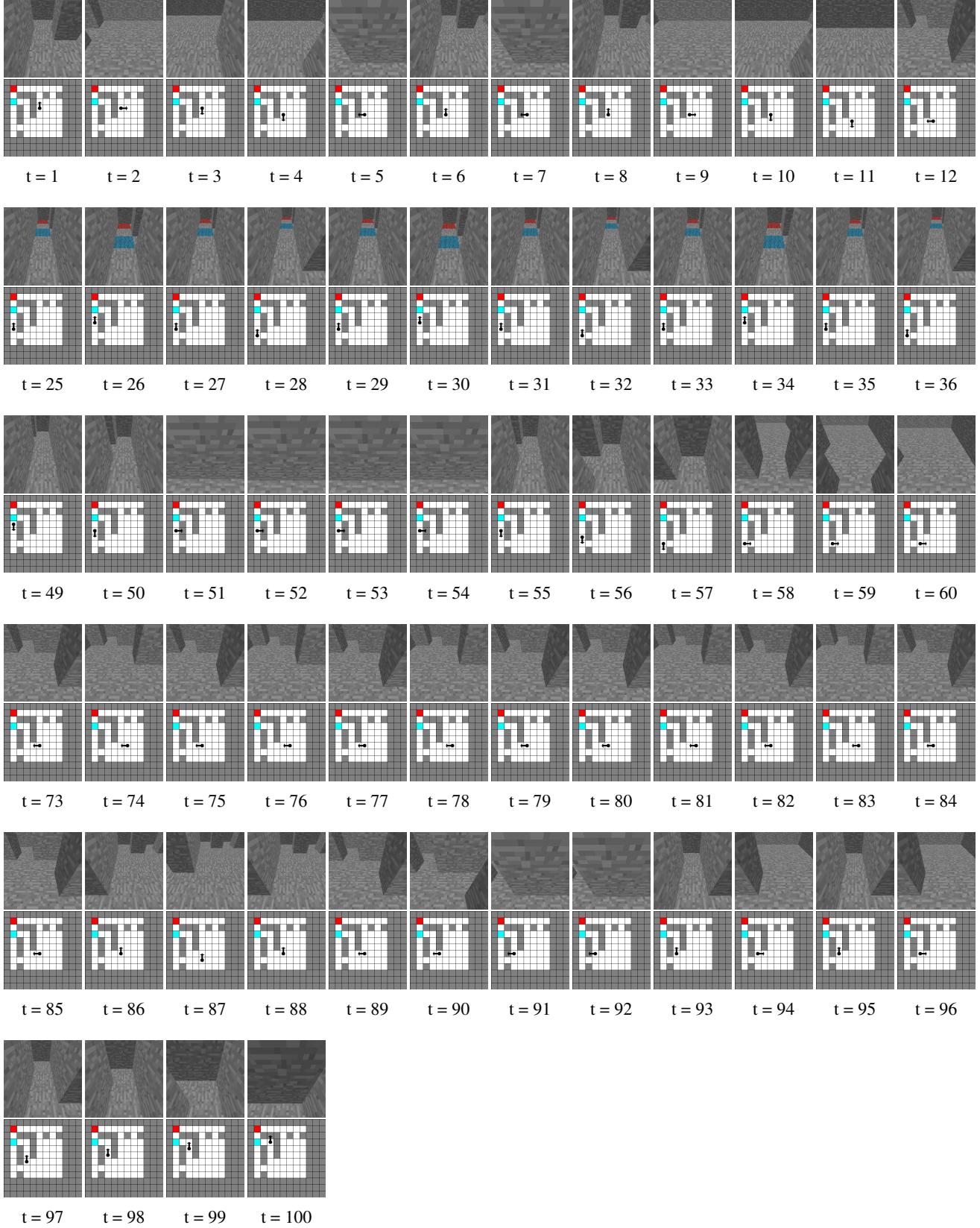


Figure 18. FRMQN’s play on an unseen and larger random maze with Sequential Goals task. At $t=25$, the agent finds both the red and blue blocks. However, visiting the red block (the first goal in the task) by following the corridor found would lead to visiting the blue block, and result in a negative reward. Thus, the agent attempts to search for another route to reach the red goal ($t=49-100$). But, the agent fails to find another route within the time limit ($t=100$).

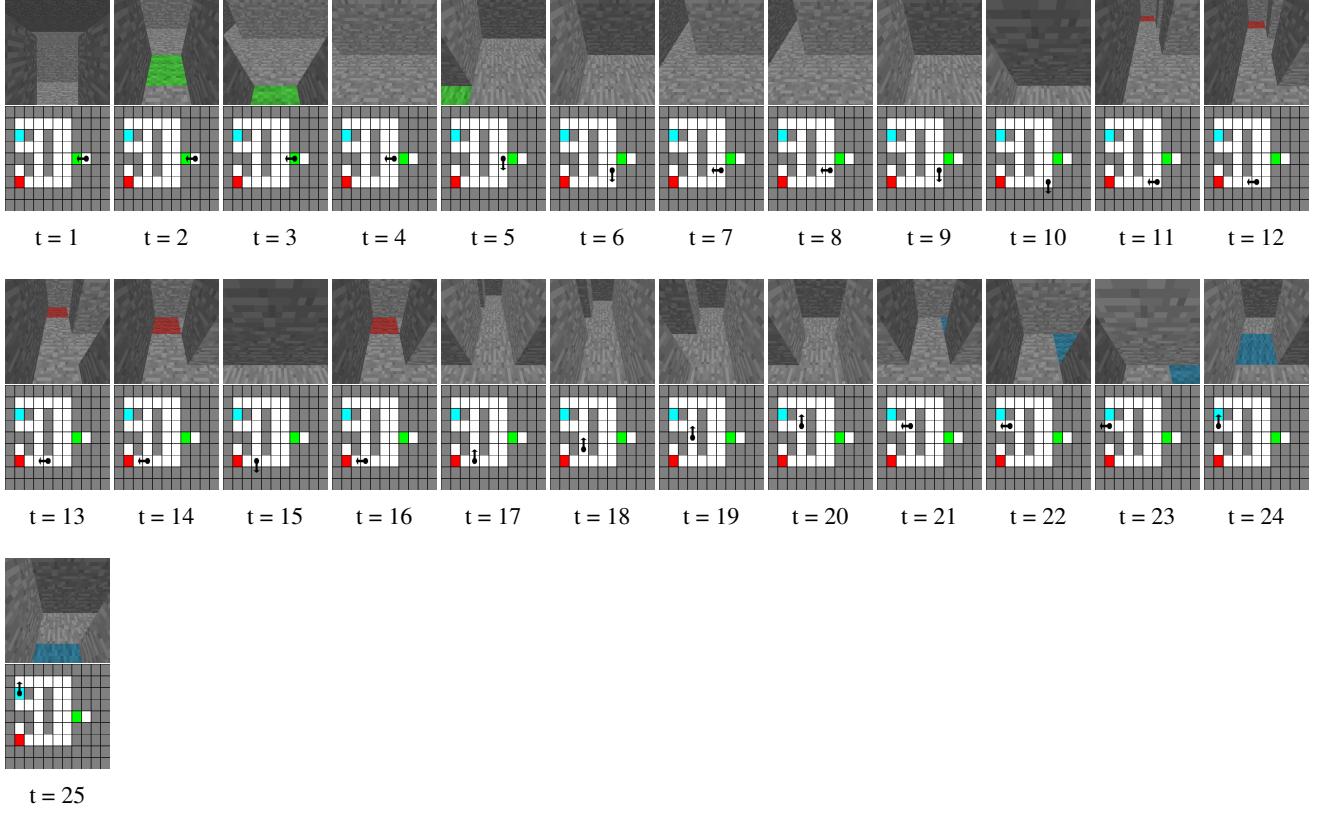


Figure 19. FRMQN’s play in a random maze with Single Goal with Indicator task. Upon seeing that the indicator is green in color ($t=2$), the agent proceeds to explore the map. During its search, it comes across a corridor with a red block ($t=11$). This is where memory helps. The agent avoids the red block (having observed that the indicator is green). From this we can infer that the agent utilizes its memory appropriately. Later, it successfully completes the task by finding and visiting the blue block ($t=22-25$).

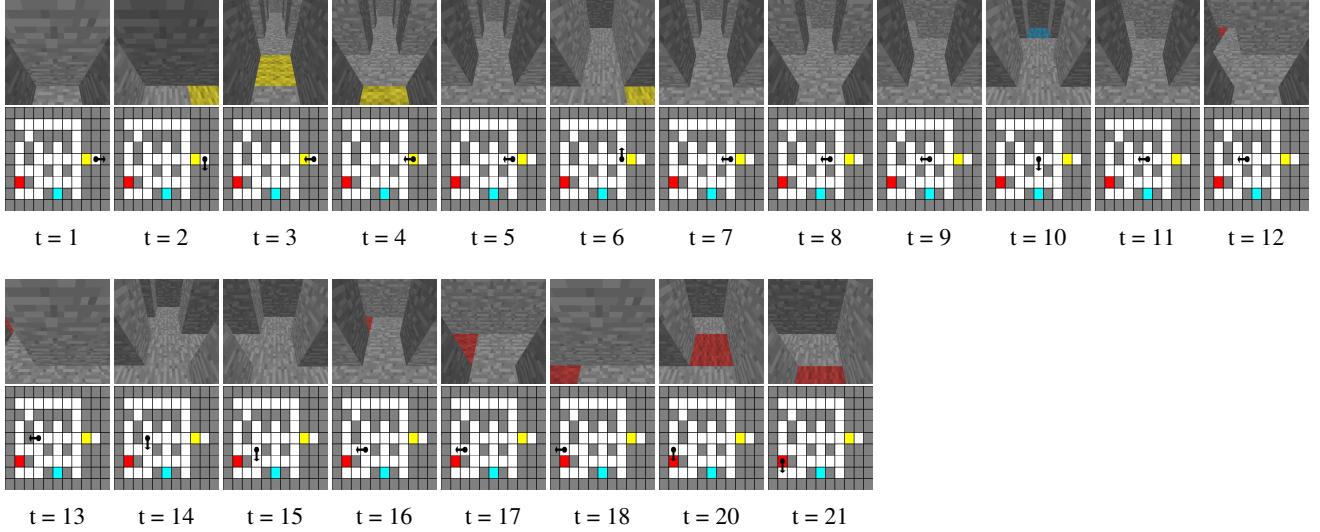


Figure 20. FRMQN’s play in a random maze with Single Goal with Indicator task. The agent visits the red block correctly, given the yellow indicator.

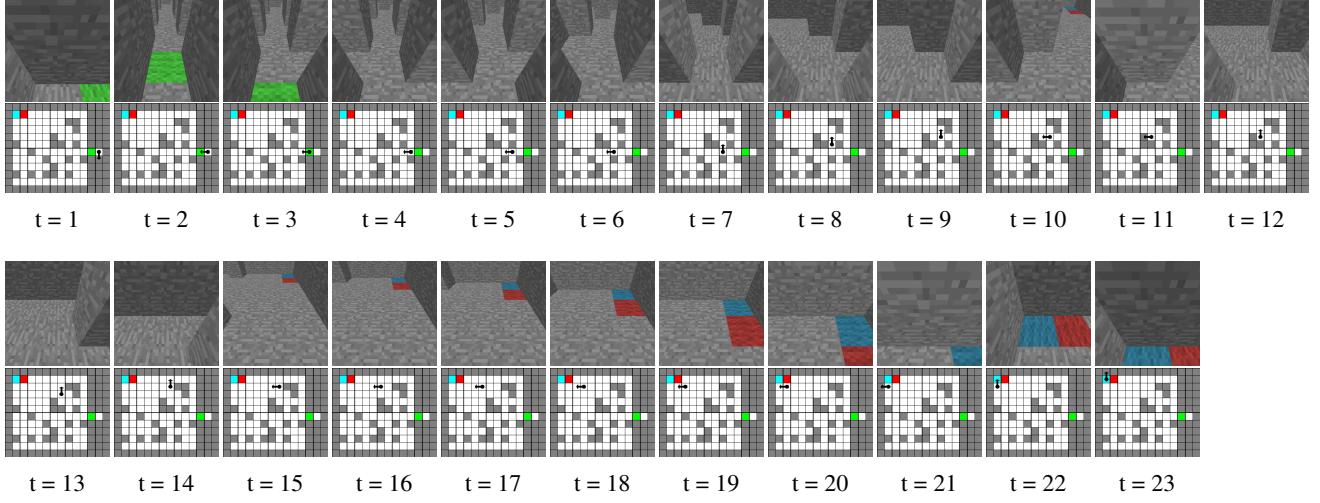


Figure 21. FRMQN’s play in an unseen and larger random maze with Single Goal with Indicator task. While the correct goal is far from the indicator, the agent is able to memorize the color of the indicator observed at the beginning ($t=3$) and visits the correct goal ($t=23$).

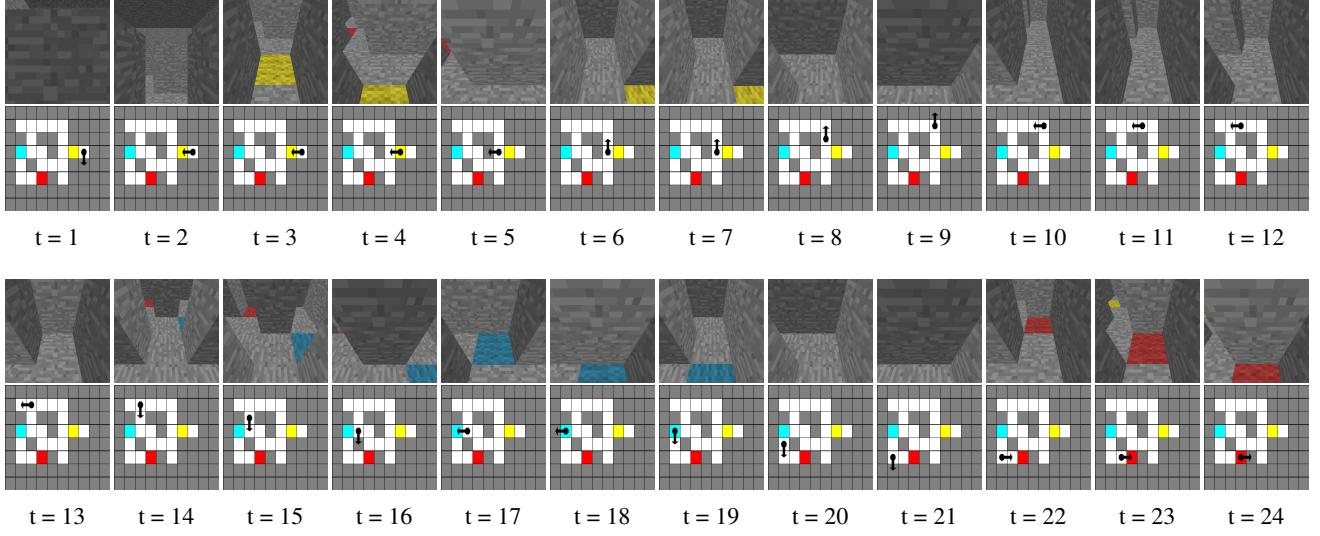


Figure 22. FRMQN’s play in a random maze with Sequential Goals with Indicator task. The agent observes that the indicator is yellow at $t=3$. The agent can see that the red block is near at $t=4$. Since the task is to first visit the blue block and then the red block if the indicator is yellow, it avoids moving towards the red block ($t=6$). The agent proceeds by turning and exploring some other corridors. Finally, it gets a glimpse of both the blue and red block ($t=14$). Using this visual observation along with retrieving from memory visual observations with the indicator present, the agent correctly goes to the blue block and then proceeds to the red block, hence completing the task ($t=15-24$).



Figure 23. FRMQN’s play in an unseen and larger random maze with Sequential Goals with Indicator task. The agent visits the red block first ($t=19$), given the green indicator ($t=3$). The agent looks for the blue block which is far apart from the red block ($t=19-47$). Once the agent finds the blue block, it finishes the task by completing the sequence ($t=49$).

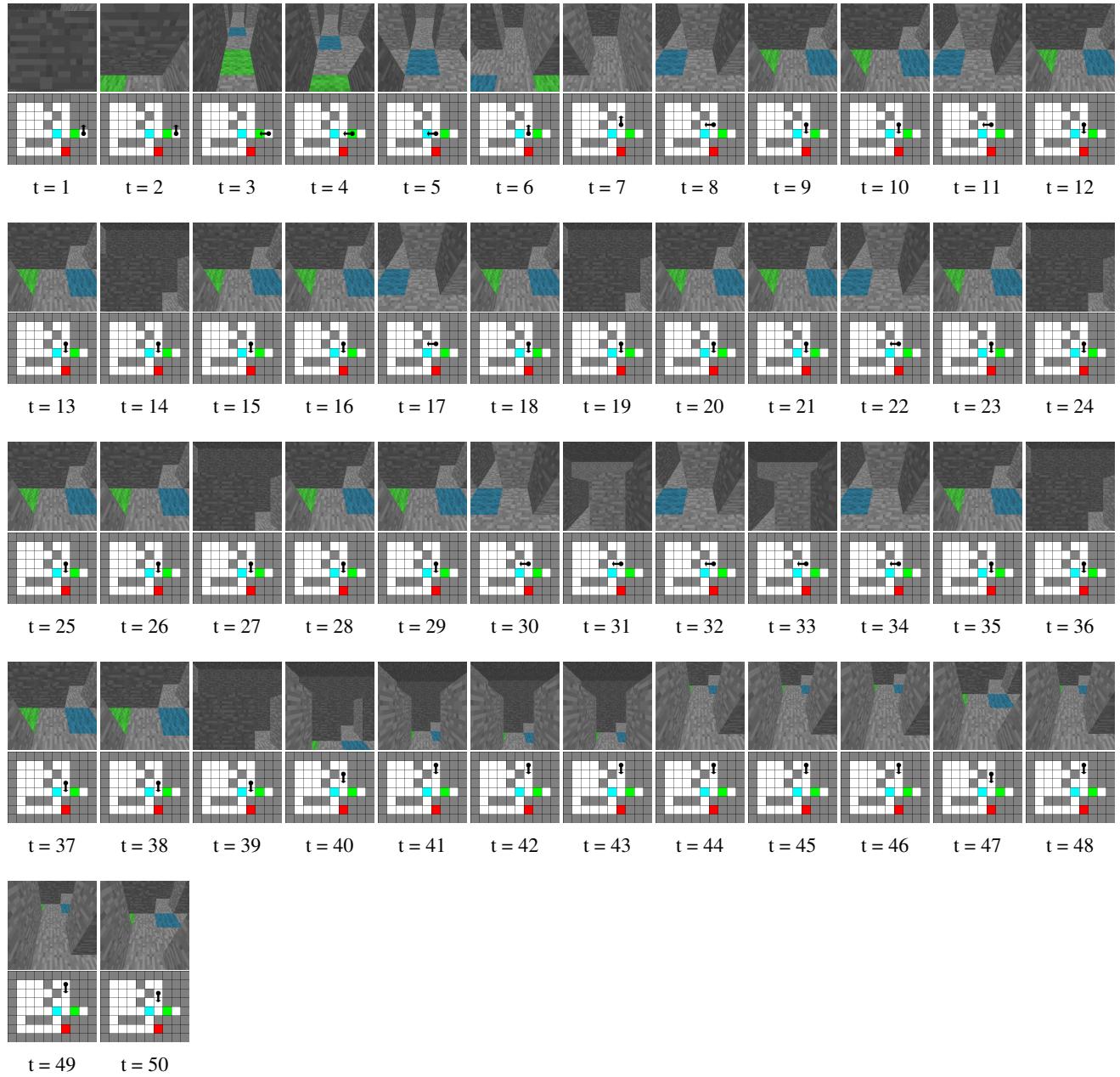


Figure 24. FRMQN’s play in a random maze with Sequential Goals with Indicator task. Given that the indicator is green, the agent has to visit the red block first and the blue block later. For this reason, the agent tries to avoid the blue block and search for another route to the red block ($t=4-50$). However, since there is no path to the red block (that avoids the blue block), the agent keeps searching until the episode terminates.