CS 221: Artificial Intelligence
Fall 2019
p-final

Russell Tran and Lydia Chan, {tranrl, lchan528}@stanford.edu

INTRODUCTION

In this paper, we train a reinforcement learning agent to play the video game Minecraft with a particular behavior of our choosing. Minecraft is a "sandbox" video game whose virtual world is composed of basic physics (e.g., gravity) and a voxel-based environment (cubes in 3-dimensional space). The virtual world is meant to mimic the appearance and layout of biomes on Earth, with rivers, forests, valleys, grasslands, deserts, tundras, etc., and players are free to manipulate the environment by breaking or placing blocks into each voxel however they see fit. The Minecraft game has no set objectives or point-scoring system by design, and so players typically interact with the game with their own human constructs in mind, such as "dig a pit", or "build a house." Given this context, we sought to teach our agent to search for bodies of water in the environment by letting the agent traverse through the space.

Our motivation for this project comes from the genuine need in society for autonomous robots and drones to perform search and rescue missions of people in hazardous and unpredictable environments such as in the case of fires, earthquakes, hurricanes, etc., where it is more cost effective and safer for droids to attempt the rescue than humans. However, successful navigation of these spaces requires research to make such robots to be extremely sensitive and adaptable to their environments. We think running simulations in Minecraft is relevant to this aim.

MINECRAFT, MALMO, AND MINERL

Infrastructure Overview

The historical progression of AI agents succeeding in the context of games with human or superhuman performance can be characterized as starting with solving narrowly-defined tasks in predictable environments (such as tic-tac-toe or chess) toward solving broadly-defined tasks in more complicated environments (Go, Atari games, League of Legends), where the state space is orders of magnitude larger, the player is responsible for an increasingly multivariate set of actions and controls, and the outcomes are nondeterministic. The implicit understanding in the field of artificial intelligence is that mastery by agents of increasingly complex games will translate as technologies which offer high utility in real-world applications, from electric grid optimization to improved dexterity of factory robots to self-driving cars. The progression toward Artificial General Intelligence is generally thought to be paved by having agents play in environments which are better and better representations of the real world.

This broad, overarching narrative is relevant to our choice to embrace Minecraft as the virtual environment for our research project. Szlam et al. (2019) [9] motivate AI research involving Minecraft with three reasons: the wider task space, building synergies between ML and non-ML forms of AI, and the opportunity for richer systems of training by humans. These reasons largely stem from the fact that the game has no set objective, so all tasks in the game tend to be more nebulous than tasks in other games in that their representation relies almost entirely on human language and ideas. Because many tasks can be defined within the same environment for Minecraft, Oh et al. (2016) [7] and Tessler et al. (2016) [10] believe Minecraft is the ideal platform for transfer learning. To some, Minecraft represents the next frontier for game mastery by artificial agents. In the past year, Minecraft has become increasingly popular as a platform for research in artificial intelligence, with at least 7 papers involving its use published between 2018 and 2019. For instance, Alaniz [1] used Monte Carlo Tree Search to teach an agent to place blocks at desirable locations; Romac and Beraud (2019) [8] have demonstrated the superiority of Deep-Q Recurrent Neural Networks over Deep Q-Learning for training Minecraft agents not to fall off a cliff; and Melnik (2019) have incorporated Semantic Modeling into training an agent to chase chickens in the game.

The two existing open source frameworks for simulating Minecraft and providing an API for an AI agent are MALMO [6] and CraftAssist [3], created by Microsoft and Facebook respectively (Johnson et al. 2016). The main difference is that CraftAssist has NLP infrastructure for converting text sentences into actions, whereas Malmo is solely a simulator. Udagawa et al. (2016) [11] have used reinforcement learning on the Malmo platform to train an agent to fight zombies. Gray et al. (2019) from the CraftAssist project have devised an agent to store symbolic memory and build circles and houses in response to conversational text-based commands. Finally, Guss et al. (2019) have built some datasets of task-completion for use in reinforcement learning through the MineRL project [5]. The MineRL project is in fact another API platform built on top of Malmo which integrates Malmo with OpenAI's gym environment system. For our project, we essentially made use of MineRL, which is built on top of Malmo, which is built on top of Minecraft.

Infrastructure Implementation

Because our infrastructure was dependent on at least three platforms abstracted on top of each other, we spent several weeks resolving dependency issues for both Windows and Mac OS. During that time frame, we also assessed CraftAssist [3], Malmo on its own, and MineRL and compared the usability and robustness of all three. We concluded that MineRL was the best-documented and most robust package for our purposes and would recommend it for any other RL research projects in Minecraft. This time investment paid off, as once the infrastructure was in place it became very convenient to simulate Minecraft in OpenAI gym, both in terms of taking actions in the environment and capturing sensory data to our specifications.

Approach: Reinforcement Learning

Challenges and Goals

The goal of our project is to build an agent that can find a source of water in an unknown environment, which we represent through using a variety of different Minecraft worlds.

The challenges in building such a system include setting up an environment to simulate this problem, both for Minecraft and for the machine learning framework and then using artificial intelligence to solve the problem.

First, we need to set up different types of Minecraft worlds and set up an interface through which the agent can interact with each of these worlds. In setting up the Minecraft worlds, we had to design the different classes of environments–basic, sparse, and dense–and tune the rewards for the environments. A large challenge in designing this system was in setting up an environment in which the agent can interact with the different Minecraft worlds. Fortunately, we were able to build upon previous work by MALMO and MineRL and interface with the Minecraft world through OpenAI Gym. However, installing and resolving the necessary dependencies proved a much more difficult process than we had imagined, as some of the dependencies were outdated or incompatible with our operating systems or other packages.

Overall, in solving our search problem, our aim was to capture visual cues in the Minecraft world that evidenced the presence, or nearby presence, of water. To this end, our problem involved computer vision, as we were utilizing visual input such as screenshots of the Minecraft world in order to search for these potential features. In the same vein, another challenge was extracting the features from the input, which we tried through two different approaches. Then, our agent needed to obtain observations and implement actions to interact with the world.

Model

Our task is to search for a feature in an unknown environment, which are represented by a source of water and a Minecraft world, respectively. Because the search-based nature of this problem, we decided to model the task as a Markov decision process (MDP).

In particular, we choose a MDP model because there are no explicitly defined transition probabilities nor reward functions. This behavior is inherent to both the design of Minecraft, which lacks a predefined objective, and the similarly open-ended definition of our search task, which is to be generalized for a wide variety of scenarios.

We shall discuss the aspects of our MDP model: states, starting state, end state, actions, and rewards.

States The states of the model are color screenshots of the Minecraft world, or 64 pixel by 64 pixel arrays of RGB values.

In order to access these states, we have set up an environment with a camera, whose observations the agent can access. Also, we are able to view these pseudo-continuous states as a video while the agent is running.

Starting State The starting state is the image array that the agent receives upon starting, or spawning, in the Minecraft world. This represents the first view of the environment.

Since we are able to contorl the spawning position and direction of the agent, the starting state is consistent throughout all the environments and algorithms.

End State The end state is the image array at the instant that the episode, which represents an attempt of the search task, ends.

Based on our definition of the environment, each episode, or trial in the experiment, will end when the agent touches water or when the step limit, 6000 steps, has been reached. Thus, the image array at this instant is the end state.

Actions The actions of the model, which represent the possible ways that the agent can transition between states, are analogous to how a human player can move through a Minecraft world. Hence, the possible actions are ”move forward”, ”turn left”, and ”turn right”, with the option to ”jump” for ”move forward”.

Rewards The rewards represent a means through which the agent receives feedback from the environment.

Water and flowing water are two common sources of water in the Minecraft world, so we reward the agent for completing the search task. To this end, the agent receives a large positive reward (+3000) for touching a block of water or flowing water.

At the same time, the environment gives a negative reward of -1 per command by the agent, or per action.

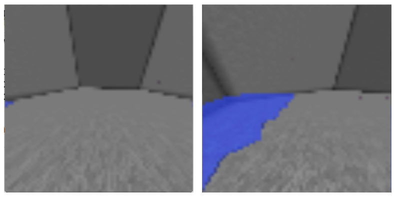
In Data and Experiments, we will discuss our reward-tuning process, from which we have defined the final reward as +3000 and the continuous reward of -1.

Environments

The main classes of environments that we used as our dataset are Basic, Sparse, and Dense environment. Through designing these three types of world, we addressed the challenge of setting up different Minecraft environment. In order for the agent to interact with these worlds, we built upon infrastructure from MALMO [6] and MineRL [5] to interface with the observation and action spaces.

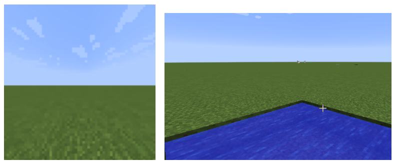
Basic Environment

The basic environment is a medium-sized stone box with tall walls. It contains water or flowing water in one corner, and the agent starts facing the wall in the opposite corner.



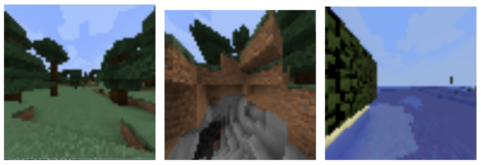
Sparse Environment

The sparse environment is a super flat grassy plains with water lakes. The agent starts at the origin of the world, and there are at least two sources of water approximately 60 units from the agent's starting point.



Dense Environment

The dense environment is a more realistic, or typical Minecraft, world with a variety of different biomes from forests to deserts to oceans to tundras and caves. There are many different sources of water, such as lakes and oceans.



Algorithms

First, to define the scope of our project, we defined and compared a baseline and oracle.

Then, we defined and our search problem. To this end, we modeled our search task (described in the Model Section) as a Markov decision process (MDP). We chose a MDP model because it would allow the agent to interact naturally with the environment based on computer vision, so we defined the frames of visual input to be the states. In addition, actions in a MDP allow for a logical transition between the different states.

Our approach to solving this problem is Reinforcement Learning. To this end, we implemented a Baseline, Q-learning, and Deep Q-learning.

To solve the search problem, we then implemented as our advanced method a Q-learning algorithm.

To improve the performance of our approach, we used Deep Q-learning with a replay buffer.

Preliminary Baseline and Oracle

First we implemented a baseline and an oracle to define the scope of the project.

For the baseline, we implemented a basic search algorithm based on breadth first search. The agent traveled in a rect-angular spiral pattern, 1 block wide, starting at its initial spawn point and facing the initial direction. It continued this trajectory until it touched a block of water.

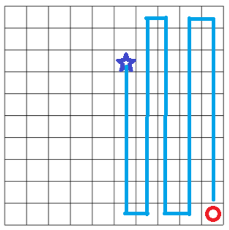
For the oracle, we had a human player find water. They explored the world and, through viewing the Minecraft world, found a source of water.

We ran the oracle and baseline for three trials, and we averaged the time it took for each to complete the search task. The baseline successfully completed the search task in 2 minutes and 13 seconds, while the oracle succeeded in 5-6 seconds. To this end, we have a large gap between the baseline and oracle performance. Thus, our search task is good, and we gained insights that our baseline implementation should be a basic search algorithm and that our approach should rely on visual input about the world, like our oracle.

Baseline

In the later stages of the project, we implemented a baseline to serve as a basis of comparison for the Q-learning and Deep Q-learning algorithms.

For this baseline, the agent traverses the world in a snake-like. First, it moves predetermined number of steps forward, turns counterclockwise (left), moves forward one step, turns counterclockwise, move the predetermined number of steps forward, turns clockwise, moves forward one step, turns clockwise, and then repeats. The figure below illustrates this algorithm.



To this end, the baseline provides the lower bound of performance that we would expect from our algorithm, as the baseline is a systematic search of the environment. In addition, this algorithm is not influenced, or dependent upon, the features of the environment, and thus offers a brute-force and unintelligent approach to the search problem.

Q-Learning

We decided to use Reinforcement Learning, and in particular Q-learning, because it fits our goal of the agent choosing particular actions to maximize its rewards. To this end, it would help solve our challenge of completing a search task in an unknown environment.

As a framework, Q-learning allows us to explore the unfamiliar environment through actions and states. More importantly, the states in the model are very suitable for our approach because we are using visual input to explore the environment.

Thus, we are tackling the challenge of computer vision through our feature extractor. We have explored two different feature extractors.

First, we have implemented an identity feature extractor that examines the observation of the environment, or screenshot of the Minecraft world. Each different frame from the observation is associated with an action, and thus represents the policy for each state.

In addition to this naive approach, we have tried to make our feature extractor more efficient by specifically looking for blue pixels, which would represent sources of water, in the visual input.

Thus, we are able to learn from our visual input about the environment and influence current and future actions that explore the environment. As a result, we are able to use this visual memory of the environment to guide the search for a source of water.

However, the downsides to these two feature extractor approaches is that they are memory- and time-expensive. For the identity extractor, it matches each unique visual input frame to a specific action, and it stores these matches in a buffer. As a result, this buffer contains a lot of data, and in fact, while running Q-learning on the basic environment, the algorithm actually terminated on the 70th episode due to the overflow of memory. In addition, this method is time-expensive because of the time needed to associate each frame with an action and then attempt to match visual input frames with those in the buffer.

For the second feature extractor approach, it is also computationally-expensive because it would require examining all of the 64x64 pixels of each input frame for blue pixels.

Thus, Q-learning provides a good start to harnessing computer vision to explore an unknown environment and solve the search task of finding a source of water. To this end, it also tackles the challenge of capturing visual cues from visual input to search for particular features. However, it is slow and uses a lot of memory, especially in the feature extractor.

From the algorithmic standpoint, we start with an initial state and estimate the Q-value for possible actions that we can use. Then, we explore different actions and their associated new states by adding these possibilities to a buffer containing past and already-explored actions and their associated rewards. We continue this process until we have converged until a solution. We define converging as having a collection of rewards whose overall averaged rewards is above a predefined threshold. If we have not converged, we sample other possible actions, and update the estimate for Q-values. Ultimately, we return a path when we have converged upon a solution.

In addition, to tackle the challenge of the agent interacting with the Minecraft world, we have built upon infrastructure by MineRL [4]. We are using wrappers to convert actions by the agent into actions for Minecraft and observations from the Minecraft environment into visual inputs for the agent. Through this methods, we are able to interface with the environment successfully.

Deep Q-Learning

To improve the performance of the Q-learning algorithm, we implemented Deep Q-learning (DQN), and we also added a replay buffer.

The Q-learning algorithm is computationally-, time-, and memory-expensive, especially with regards to its feature extractor. As a result, we consider DQN, in which we use a convolutional neural network as a feature extractor.

Like with Q-learning, DQN takes in visual input from the Minecraft world as a 64 by 64 array of RGB values of the pixels in the frame. It differs in that it runs this array through a deep neural network with three convolutional layers and one hidden layer. It then uses this output to choose a new action to explore.

We also created a replay buffer to store 30,000 transitions, and for each step, we sample from this replay buffer to help minimize the error in Bellman's equation.

Thus, DQN improves upon the Q-learning in that it more efficiently extracts features from the visual input, through the use of deep neural networks, and thus it much more time-efficient.

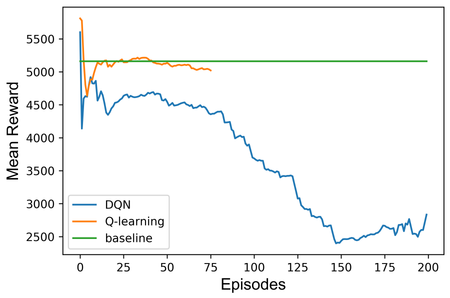
Results

The link to the code and data for our project is at https://github.com/Russell-Tran/CS221_find_water.

To test the search problem of finding a source of water in an unknown environment in Minecraft, we ran different experiments in different types of environments to compare the performance of the algorithms.

The experiments that we ran were on the Basic environment (presented in our p-progress and p-poster), the Sparse environment and the Dense environment. The metrics that we used to compare the three different algorithms, Baseline, Q-learning, and Deep Q-learning, are the Mean Episode Reward and Total Steps.

We expected that the DQN performs best, then Q-learning, and lastly our Baseline, which is supposed to represent the lower bound of algorithm performance. However, for the Basic environment, we saw that the DQN did not, in fact, outperform the Baseline and Q-learning. In fact, the mean episode reward is shown to decrease and possibly converge for both Q-learning and DQN:

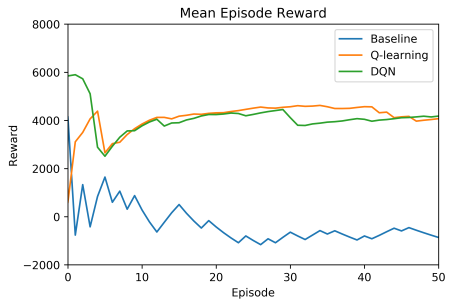


Comparison of Mean Rewards over Episodes for Basic Environment.

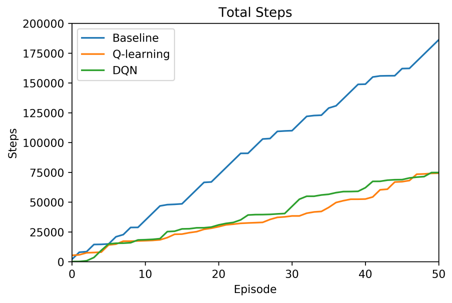
To this end, the Basic environment results may have been due to the uniformity of the environment: a stone box with a stone floor and four stone walls. The Baseline systematically traverses the box, so it will reach the water in the corner at some point. However, Q-learning and DQN are exploring the environment based on visual cues, so it is likely that the lack of unique features in the environment would be unhelpful and possibly misleading. In this vein, the performance of Baseline over the other two algorithms in the Basic environment is initially surprising, but it is logical considering the lack of unique visual cues and features in the environment.

For the Dense environment, we see that DQN and Q-learning both perform significantly better than the Baseline, as expected: the former two have higher mean episode rewards than the latter. In addition, we see that the DQN performs slightly better than Q-learning, and during run time, we noticed that the DQN ran approximately 5 times faster than Q-learning. To this end, we see that DQN performs better and more efficiently than Q-learning, as expected.

The Mean Episode Reward is a metric measuring the accumulated average of rewards. With this metric, we can see how the reward received by the agent changes with each episode. We would expect it to increase, as the agent learns more about the environment and, as we expect, increases its rewards. The Mean Episode Reward metric is more informative than simply Reward per Episode metric, shown below, because it demonstrates the improvement of the agent with time and thus offers a more holistic picture:

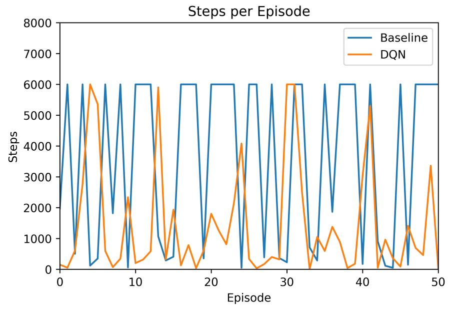


Mean Episode Reward per Episode for Dense Environment



Reward per Episode for Dense Environment

Other metrics that we collected are the Reward per Episode metric and Time Steps per Episode metric. These metrics have similar results among the different algorithms, and they demonstrate the variance among different episodes for the same algorithm. The number of steps per episode is capped at 6000 steps, in order to keep the run times of the algorithms reasonably long.



Time Steps per Episode in Dense Environment

Error Analysis and Literature Review

We had hypothesized that for any given environment, we would expect DQN to perform better than our baseline and better than our simple Q-learning approach. However, in the Basic environment, we discovered that our baseline performed best, and DQN performed worst of all. Given that we thought the Basic environment was the easiest to solve, this was not an intuitive result, so we investigated potential sources of error. From this review we found that water occasionally “froze” into an ice block in our Basic Minecraft environment, making it impossible for an agent to successfully touch water, so we modified environment settings to ensure this would not happen again. Furthermore, we tweaked the reward function to strike a reasonable balance between the amount of reward to give the agent for successfully touching a water block and the amount of punishment to give the agent for each additional action it would take: a reward that was too high relative to the punishment might render the punishment negligible, hindering our agent from learning to take the most efficient route instead of wasting time; and a reward that was too low might prevent the reward from having any meaning at all. This is how we devised a reward of +3000 upon successful completion and a punishment of -1 for each additional step expended. Ultimately, this did not change the consistency of our results with regards to our Basic environment.

Though this result was not intuitive, it is actually consistent with the literature. Guss et al. (2019) found that a DQN agent searching for metal blocks in Minecraft performed worse than random movement in sparse environments, but in fact outperformed random movement in dense environments. The understanding from this is that the more complex environment has a greater diversity of frames from which the DQN can make distinctions from trees, terrain, grass, etc., whereas these distinctions cannot be made by the neural network in an extremely homogenous environment such as Basic or Sparse. This explains why, for instance, both Q-learning and DQN had a tendency in the Basic environment to actively run up against a gray corner, which is of course dramatically less efficient than a baseline which systematically traverses every area. Furthermore, our Basic environment was constructed sufficiently small such that the blind systematic policy was generally superior to any other policy that might entail exploration.

This behavior is a fundamental drawback to DQN approaches in general, even beyond the context of Minecraft, known as perceptual aliasing. Zaval & Gureckis (2010) [12] define this as “arising in situations where multiple, distinct states of the world give rise to the same percept.” Frazier and Riedl (2019) [2] have also recently demonstrated the perceptual aliasing phenomenon in DQN approaches used for Minecraft, where they attempted to alleviate it with human action advice. They emphasize the fact that the textures (pixel styles) on blocks of the same type in Minecraft are always identical, which worsens the problem.

From this, we have concluded that our experiments in the Sparse and Dense environments, beyond the Basic environment, did not have any other significant sources of error. This is because our results continued to be consistent with the literature in that our own Dense environment did demonstrate a significantly better performance of DQN over Q-learning and the baseline.

Conclusion and future work

There is still significant research to be done for artificial intelligence in the context of Minecraft, but even in the context there is work to be done. For instance, we hope to:

1) create more complex and realistic scenarios. For example, we might create an Extremely-Dense environment, or environments that require higher reasoning, such as mazes. Then, we plan to use variations of these worlds for learning and work towards more and more complex environments.
2) strategically vary the rewards given to the agent. In our preliminary results, we have experimented with different reward values and types, and we plan to extend this in a more strategic and systematic manner.
3) implement improved training algorithms, such as those described by Frazier & Riedl (2019) and Romac & Beraud (2019), as it is clear that perceptual aliasing is a major hindrance to DQN. With improved training algorithms which show a higher success rate, we would hope to be able to observed learned heuristics by the agent which were somewhat present here with DQN but did not fully manifest, such as the clear ability for the agent to relate the color blue to water, to distinguish the blue of the sky from the blue of the bodies of water, to recognize sand and other distinguishing features of bodies of water as attractive, and to recognize the general strategy that downhill movement as opposed to uphill movement increases the chances of finding water.

References

[1] S. Alaniz. Deep reinforcement learning with model learning and monte carlo tree search in minecraft, 2018.

[2] S. Frazier and M. Riedl. Improving deep reinforcement learning in minecraft with action advice, 2019.

[3] J. Gray, K. Srinet, Y. Jernite, H. Yu, Z. Chen, D. Guo, S. Goyal, C. L. Zitnick, and A. Szlam. Craftassist: A framework for dialogue-enabled interactive agents, 2019.

[4] W. H. Guss, C. Codel, K. Hofmann, B. Houghton, N. Kuno, S. Milani, S. Mohanty, D. P. Liebana, R. Salakhutdinov, N. Topin, M. Veloso, and P. Wang. The minerl competition on sample efficient reinforcement learning using human priors, 2019.

[5] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. Minerl: A large-scale dataset of minecraft demonstrations, 2019.

[6] M. Johnson, K. Hofmann, T. Hutton, D. Bignell, and K. Hofmann. The malmo platform for artificial intelligence experimentation. In *25th International Joint Conference on Artificial Intelligence (IJCAI-16)*. AAAI - Association for the Advancement of Artificial Intelligence, July 2016.

[7] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft, 2016.

[8] C. Romac and V. Béraud. Deep recurrent q-learning vs deep q-learning on a simple partially observable markov decision process with minecraft, 2019.

[9] A. Szlam, J. Gray, K. Srinet, Y. Jernite, A. Joulin, G. Synnaeve, D. Kiela, H. Yu, Z. Chen, S. Goyal, D. Guo, D. Rothermel, C. L. Zitnick, and J. Weston. Why build an assistant in minecraft?, 2019.

[10] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft, 2016.

[11] H. Udagawa, T. Narasimhan, and S.-Y. Lee. Fighting zombies in minecraft with deep reinforcement learning. 2016.

[12] L. Zaval and T. M. Gureckis. The impact of perceptual aliasing on exploration and learning in a dynamic decision making task. 2010.