What Happens To ZORK When RL Meets DL

Akshay Sharma(as6429)[1] **Riya Gupta(rg3332)**[1] **Sairam Satwik Kondamudi(sk4824)**[1]
Sreeharsha Varma Tinnanuri(st3364)[1]

Abstract

Text games or popularly known as Interactive Fiction Games have been quite popular for a very long time. Current reinforcement learning research trends includes exploring and improving these text games which comes with lots of challenges like partial-observability and large action-state space. In this project, we explore the idea of playing text-based games by training a Reinforcement Learning (RL) agent in the textworld environment. We perform our experiments on the game Zork, which is a fantasy based text game. We investigate Deep Reinforcement Learning, with the help of different RL algorithms, such as SARSA and Q-Learning along with different RNN architectures. We also try to draw a comparison between the different agents and their performance. We show that the DDQN agent performs better than an agent that has picked an action randomly. Additionally, we present a brief finding on different values of ϵ and the effect on our game scores when the agent discovers new items.

Keywords: Deep Q-Network; Double Q-Learning; Multi-agent Systems; Reinforcement Learning, Natural Language Processing, Interactive Fiction Games, Policy Iteration, Online Policy, Offline Policy

1. Introduction

Reinforcement Learning deals with the agent, their actions and interactions with the environment such as to maximise the reward and decide next actions (Sutton & Barto, 2018). for the past couple of years, researchers have started integrating deep learning with the reinforcement learning has been an area which has been largely researched for the past couple of years for games and in fields like natural language processing and computer vision, hence adding a new dimension to the ongoing research trend. Today, numerous multi-player games such as Go (using AlphaGo Zero (Silver et al., 2016)), Chess, Atari, and Sudoku can be played successfully - in some cases to a professional degree - using deep learning based reinforcement agents. They demonstrate how we have been in training agents to optimize decision making processes in different fields.

One such field which is gaining attention lately is simulating Reinforcement Learning agents in a natural language processing environment or popularly known as Interactive Fiction Games, also called text-games. Text games encompass a plethora of challenges since the deciding factor for the decision making process is not the reaction time; instead it is the understanding of the semantics and common-sense about the language (Luketina et al., 2019). Various algorithms and approaches have been used in order to deal with challenges - deep reinforcement learning (Narasimhan et al., 2015) being the prominent one - to jointly learn state representations and action policies using game rewards as feedback. Due to their partial observability and characteristic reward sparsity, it is a challenging task to extract complete information about the state using descriptive texts. In simpler terms, both action and observation space is language when it comes to text-games and are both combinatorial and compositional which makes learning difficult. Figure 1 shows the basic working of the text games.

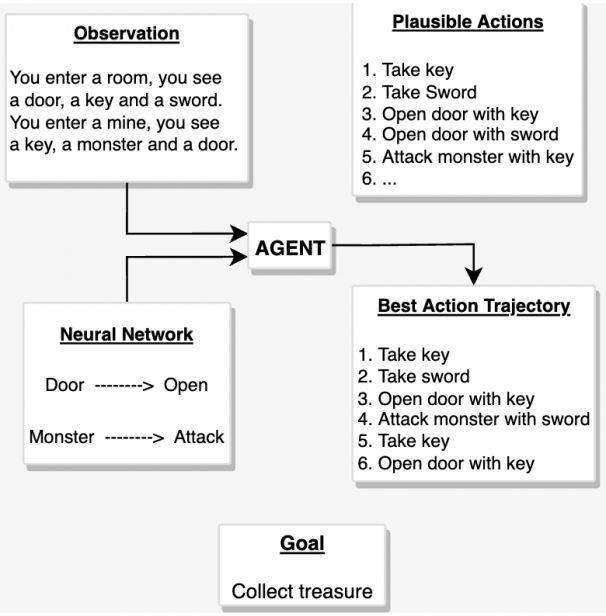

Figure 1. Representation of text-game

Figure 2. Screenshot examples of action for Text-World

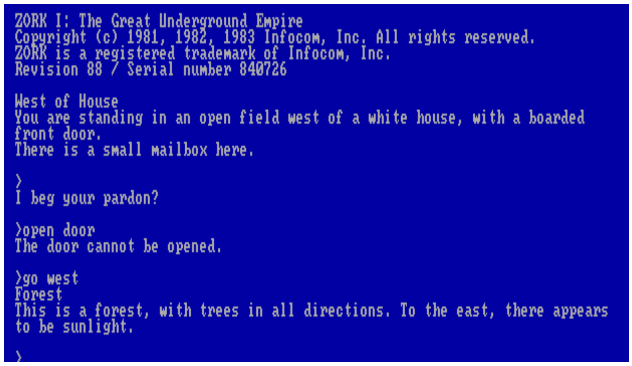

Figure 3. Screenshot examples of action - reaction for Zork

Engines such as Textworld(Côté et al., 2019) and Jericho (Hausknecht et al., 2020) are popular generation-cum-simulation tools for training agents that observe and interact using the natural language interfaces. They provide an interactive shell, where the user prompts an input to a sandboxed environment (Wright et al., 2006) - cooking in the kitchen, cleaning the house - with sets of instructions or actions and their outcomes. These tools are completely generative in nature. Figure 2 and 3 shows the interactive shell with outputs from Textworld and Zork.

For the purpose of our discussion, we are training agents to play a fictional text game called Zork (inf) using the textworld engine. Zork is a late 1970s text game which can handle a variety of simple (”attack monster”) and complex (”attack the monster with sword”) commands, making it well-versed in language parsing. We train RL agents that play this game using DDQN(Fedus et al., 2020), which is a deep learning based approach to approximate the state function with the help of stacked neural networks.

Our contribution to this work is threefold:

- Compare different reinforcement learning algorithms for text-based game play and present a survey on our findings
- Compare different exploration strategies (ϵ-greedy and Thompson sampling)
- Compare different architectures and their effect on learning embedding for game state

2. Learn with Zork

Zork is a complex and interactive fiction game, with maps above and below the ground for achieving certain goals. Each map has sets of rooms and interact-able objects. Players can also move in certain directions from these rooms making it more complex in nature as the action-space changes from room to room.

Commands : Zork takes a limited set of commands which are divided into 3 parts :

- Basic Commands : commands related to movement or obtain information, includes commands like ”go north” or ”go east” or ”d”
- Verb-Object Commands : commands that includes verb, noun or noun phrase. It follows structure like ”open OBJ” where OBJ can be any object for example ”Open wooden door” or ”take the key”
- Verb-Object-Prep-Object Commands : these commands consists of verb and a noun phrase followed by a preposition and second noun phrase, following structure such as ”cut an OBJ with DCT”. An example of these commands could be ”Attack the monster with the Sword”

Rewards : A player receives certain rewards on finding and acquiring various objects. These are some ways in which players are awarded in Zork :

- Negative Reward : -1 for taking too many turns in the game
- Positive Rewards :
  - when a player discovers a new area/room : 2
  - exploration reward for not staying the same area for long : 0.5
  - on finding and adding the item to the player inventory : 3
  - for adding repetitive items to inventory : 0.5
- In Score Game Reward : The current score of the player depending on the best action taken

How it works? : Player’s inventory and surroundings act as a state and provide information to the agent regarding

	Count	Examples
Rooms	110	*Attic, Kitchen, Living Room*
Takeable Objects	59	*Leaflet, Jeweled Egg, Lamp*

Table 1. Statistics of Zork game (both above and below ground map)

actions which assist in taking the next decision. *Exploration* is the main part of the game where player moves around the different rooms and takes different paths to achieve the maximum reward after finding the necessary items for ending the game.

3. Definitions and Algorithms

In this section we are explaining some basic algorithms that are relevant to our project.

3.1. SARSA

State–action–reward–state–action (SARSA) is an on-policy learning algorithm which updates the Q value taking into account the possible reward received in the next time step of an episode and the discounted future reward received from the next state-action observation. The update target for SARSA is

$$Y_t = r_t + \gamma Q(s_{t+1}, a_{t+1})$$

3.2. DQN

Deep Q-Network (DQN) approximates a state-value function in a Q-Learning framework with a neural network. It is used in conjunction with experience replay, wherein episodes are stored in memory for off-policy learning and they are sampled during replay. In essence, the neural network is composed of stacked recurrent and convolutional layers which approximate the Q-function. Figure 4 shows the algorithm for DQN with experience replay

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 4. DQN algorithm with experience replay

3.3. Thompson Sampling

Thompson sampling (Russo et al., 2017) is a heuristic for choosing actions that maximizes the expected reward with respect to a randomly drawn belief,

$$a^* \sim \mathbb{E}[a|s_t, r_t, s_{t+1}]$$

3.4. Experience Replay

Experience replay(Fedus et al., 2020) is a replay memory technique which is used in RL, which acts a buffer for the agent's memory. It collects/stores the most recent transitions of the off-policy algorithms at each time step i.e $e_t = (s_t, a_t, r_t, s_{t+1})$. it stabilizes the network and overall improves the efficacy by keeping a track of all the actions, states, paths and rewards during training enabling the data to be reused multiple times.

3.5. Textworld (Côté et al., 2019)

Microsoft TextWorld is an open-source, cross platform engine that helps us simulate the Zork game. We train our agents using its interface as it provides APIs that bridge the gap between language understanding and decision making. Its interface is almost similar to OpenAI's Gym and provides two crucial functionalities - ability to update take an action using text command and receive updates (new state, reward, and inventory respectively) after taking the action.

4. Methodology

In this section, we explain the architecture and the different tweaks we made and their result comparison. We also demonstrate the end game scores on various agents.

4.1. Architecture

To model our game with deep learning, we design a framework that uses Double Deep Q network(DDQN) network, which is also known as double Q learning(van Hasselt et al., 2015). DDQN can be generalized to large action space hence can approximate large functions. In the paper(van Hasselt et al., 2015), authors share how DDQN is often optimistic in updating state-action pair values. To this end, the authors train two networks (essentially two Q-functions), where one is used to update the other. During each update, one set of weights get updated to find the greedy policy and the other weight gets updated to find out the value of that policy. The target for DDQN is :

$$Y_t = r_t + \gamma Q(s_{t+1}, \arg\max_{a \in \mathbb{A}} Q(S_{t+1}, a))$$

Here we have two models, primary model and secondary model. Training happens on the primary model whereas for prediction, second model is used. The updated weights are finally moved from main to secondary. DDQN is an

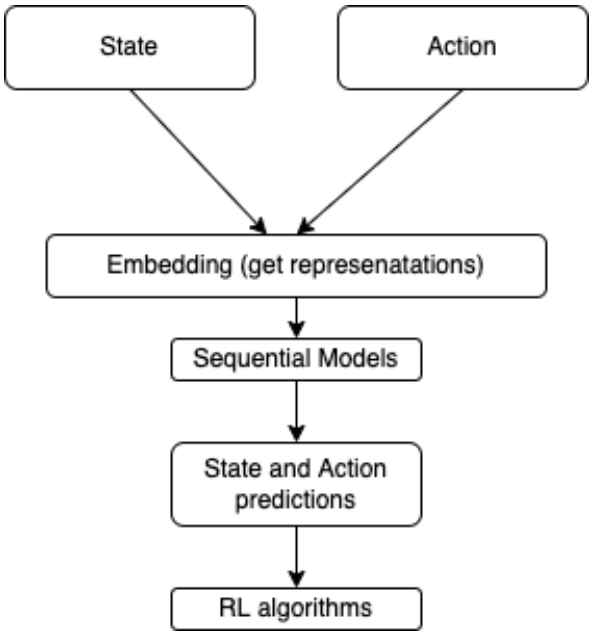

Figure 5. Architecture of the Model

alternative and better architecture than the DQN as it is more stable and combats the problem of overestimation of rewards over time. The action-space and state-space is very large in the which is also one of the main reasons to use DDQN instead of DQN for this problem statement.

We have a set of instructions which are used to interact with our agent. These instructions are in text format, which are first preprocessed using basic text processing steps like text-cleaning, tokenization, lemmatization etc. This input acts as both are state and action. We extract the features from these inputs using a shared embedding space which generates similar word representation for action and space both.

These extracted features of both action and space are further passed to RNN models due to their better ability to capture text dependencies. We use 3 different artificial recurrent architectures here on RNNs, LSTMs and GRUs and how agent learns with these three different models. The resultant models are trained during experience replay and during each episode they are used to predict best action and Q-value given the current game state. We use RMSProp as our optimizer and mean squared error as our optimisation function.

5. Results

In this section, we share results of different studies that we conducted using the DQN agent. Its worth noting that the rewards fed into the model do not reflect the actual game rewards. This is important since most text games are sparsely rewarded - the agent receives a reward only when it collects some valuable artifact. We generate modified rewards for each game iteration using simple heuristics,

- A small penalty for every move in a game
- A reward for exploring new states
- A large reward for exploring new inventory items

Unless mentioned explicitly, the term reward in our studies means this modified score.

5.1. Comparing different agents

We study how an agent trained with DDQN performs on selecting actions relevant to the game prompt and compare it with a baseline (which selects actions randomly). To this end, we train DDQN agent to play Zork 32 times with 64 rounds/episodes each game and batch size of 8. The end of game scores are shown in Figure 6. DDQN agent outperforms baseline model in every game after first few iterations. Note that we used two variants of DDQN agent - one that uses Thompson (or multinomial) sampling and the other uses ϵ-greedy approach to select the best action. Multinomial sampling, based on action probabilities, allows the agent to iterate much faster compared to finding best action by iterating over the complete action space. This is intuitive since the action space explodes exponentially as the agent discovers new states, inventory, and game entities.

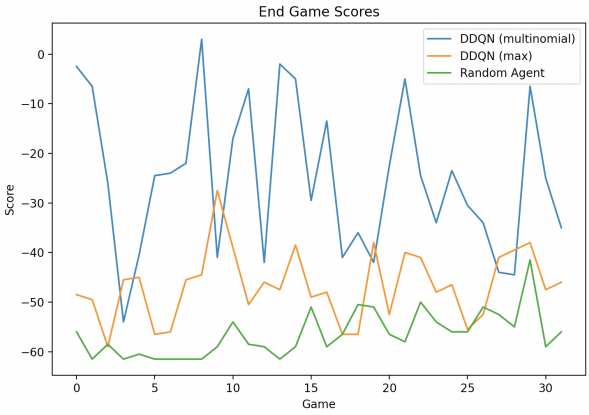

Figure 6. Comparing DDQN agents with a random agent baseline

5.2. Effect on exploration on gameplay

We compared how initial value of ϵ affected the end of game scores for a DDQN agent. For this, we trained a DDQN agent to play the Zork game - 16 games, 32 rounds each, with a batch size of 8. The agent followed an ϵ-greedy exploration strategy. The values of ϵ chosen at the start of the game were 0.4, 0.6, 0.8, and 1.0 respectively. Note that the agent decays ϵ by a factor of 0.999 exponentially

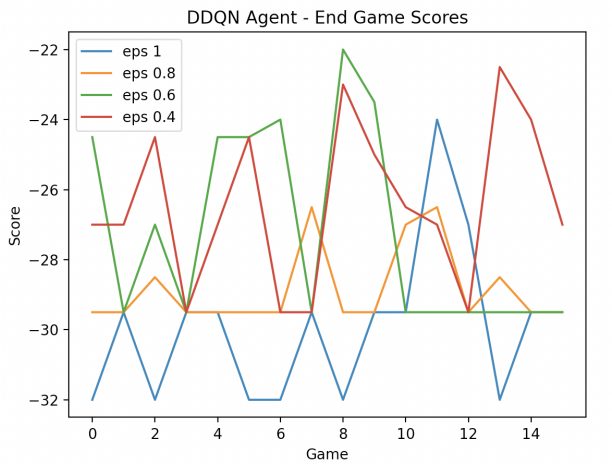

Figure 7. Effect of initial ϵ on gameplay given same decay

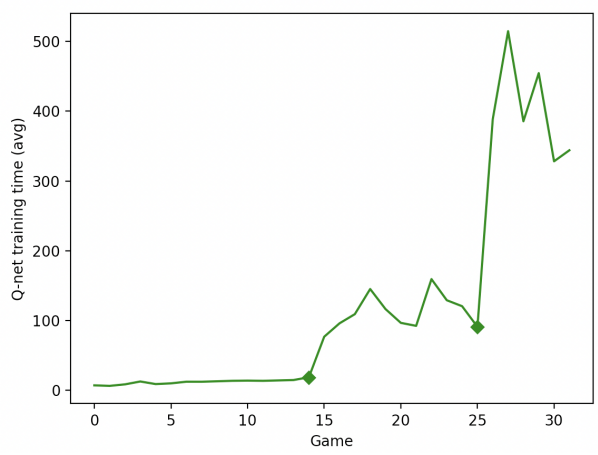

Figure 8. Effect of new discoveries on action space size

after every replay. The results are shown in Figure 7. The value of ϵ for optimal gameplay i.e. balanced exploration vs exploitation depends on the total number of times agent replays - $epsilon$ should saturate to a low value only near the end of the game. For our chosen game configuration, larger value of ϵ result in near-random gameplay and low end-of-game scores. Due to the sparsely rewarded nature of text games, it is imperative for an agent to explore new states and discover new inventory items. But since the action-space is huge for text-based games, repeated exploration often results in redundant, if not invalid, actions.

5.3. Effect of discoveries on gameplay

We study how the action space changes when the agent encounters new inventory items. We consider a simulation of 32 games with equal rounds each. Although the agent discovers new inventory items in every game, each inventory item has a similar impact on the gameplay. The results are shown in Figure 8. In games 14 and 25 (0-indexed), the agent discovers two key inventory items viz. *brass lantern* and *the wand*. Unlike other parts of inventory, these have a pronounced effect on gameplay viz.

- *brass lantern* allows agents to explore new game states via an *underground tunnel*
- *the wand* allows agents to explore new game states and conjure new inventory items

Irrespective of how they contribute to the game map exploration, both items have a marked result on the action space that the agent iterates over during every game round. This shows that inventory items in the Zork game have a non-linear impact.

5.4. Comparing different artificial recurrent neural networks

In this section we compare how different recurrent neural networks perform and capture dependencies given the game description and inventory. We train RNNs with a batch size of 64, vocab size of 1200. The inputs are the joint representation of state and action obtained from the shared embedding layer of dimension 16. The number of hidden layers we have used are 32, with a dense layer of size 8. We see that RNN performs better in the case of Zork, for 30 games. We also see that GRU has highest score peaks for some of the games. We believe if we increase the number of game runs, then the scores for GRU and RNN would be similar, whereas the random agent ends up performing worse. Figure 9 shows the comparison between the different RNN architectures that we implemented and their performance.

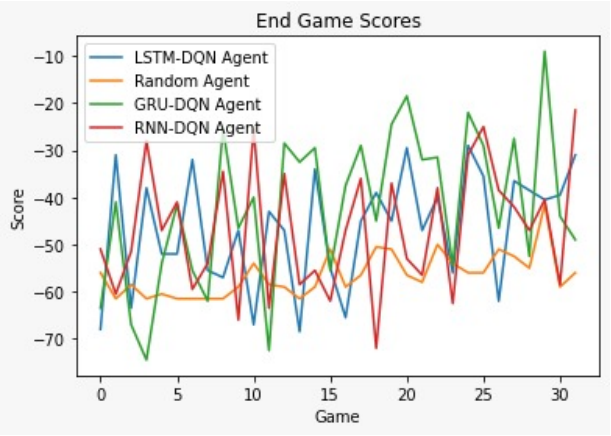

Figure 9. Comparison between the different RNN architectures

6. Hyper-parameter Complexity

We used certain hyperparameter setting and the computational resources in order to run our simulation. Table 2 presents the statistics of the machine that we used in order to train the RL text game.

We used 1 GPU to train the network. The code that we have submitted, can also be converted to the parallelized model.

Resource	Settings
GPU	NVIDIA GTX 1080 12GB
CPU	INTEL(R) CPU x86—64
Frequency	2.099GHz
OS	Ubuntu 16.04.6 LTS (Xenial)
Disk	438G

Table 2. Statistics of the machine

Table 3 presents the hyperparameters for the deep RL algorithms

Parameter	Value
ϵ (initial)	1
ϵ-decay	0.995
Vocab Size	1200
γ	0.75
α (learning rate)	0.1

Table 3. Hyper-parameter Statistics for RL algorithms

7. Conclusion and Future Work

In this project, try to teach our agent to interact with the user and take actions and return rewards using the integration of reinforcement learning and deep learning. After surveying different text games, deep reinforcement learning for Zork, which is a text game (IFG). We try to compare different strategies to play text games using reinforcement learning which includes the effect of learning on using different recurrent neural networks along with comparing the performance of DDQN with random agent.

There are a few work directions that warrant more investigation. Although similarity based heuristics (Word2Vec, word embeddings) help the agent narrow down possible combination of game entities and verbs to create an action, text-games are rife with many game entities and often game entities for an action are chosen randomly. We would like to explore how other approaches such as hierarchical tree-banks (WordNet) or complex measures in causal inference can help the agent determine a game entity to act upon given the current state.

8. Acknowledgements

We would like to acknowledge Professor Li, Professor Wang, TA Sam Fieldman and other TAs for their efforts throughout the semester. Everyone has been a great help to our group throughout the semester by answering our questions related to lectures and project.

We were allowed an extension of two days to submit this report.

Codes and relevant codes to our project can be found on this github link: What happens to Zork when RL meets DL

9. Contribution

All the 4 members of this team contributed equally for generating the results mentioned in the article and generating the report.

References

Zork. URL "https://ifdb.org/viewgame?id=0dbnusxunq7fw5ro".

medium article on zork. URL "https://medium.com/data-by-matt/deep-zork-part-1-2ea114e62e43".

Côté, M.-A., Kádár, , Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., El Asri, L., Adada, M., and et al. Textworld: A learning environment for text-based games. *Computer Games*, pp. 41–75, 2019. ISSN 1865-0937. doi: 10.1007/978-3-030-24337-1_3. URL http://dx.doi.org/10.1007/978-3-030-24337-1_3.

Fedus, W., Ramachandran, P., Agarwal, R., Bengio, Y., Larochelle, H., Rowland, M., and Dabney, W. Revisiting fundamentals of experience replay. *CoRR*, abs/2007.06700, 2020. URL https://arxiv.org/abs/2007.06700.

Hausknecht, M., Ammanabrolu, P., Côté, M.-A., and Yuan, X. Interactive fiction games: A colossal adventure, 2020.

Luketina, J., Nardelli, N., Farquhar, G., Foerster, J., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T. A survey of reinforcement learning informed by natural language. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 6309–6317. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/880. URL https://doi.org/10.24963/ijcai.2019/880.

Narasimhan, K., Kulkarni, T., and Barzilay, R. Language

understanding for text-based games using deep reinforcement learning, 2015.

Russo, D., Van Roy, B., Kazerouni, A., Osband, I., and Wen, Z. A tutorial on thompson sampling. *arXiv preprint arXiv:1707.02038*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, jan 2016. ISSN 0028-0836. doi: 10.1038/nature16961.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 0262039249.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

Wright, W., Schroh, D., Proulx, P., Skaburskis, A., and Cort, B. The sandbox for analysis - concepts and methods. volume 2, pp. 801–810, 04 2006. doi: 10.1145/1124772.1124890.

Matt. Medium article for Zork (med)