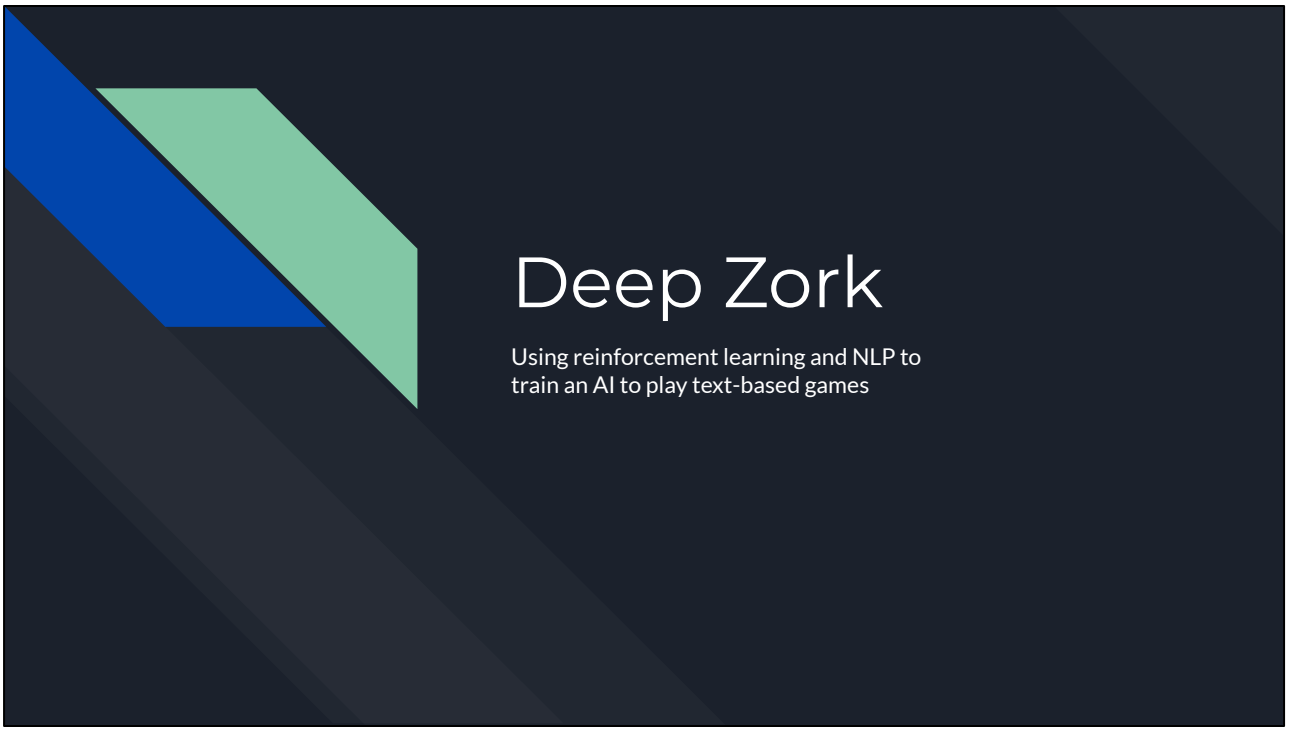
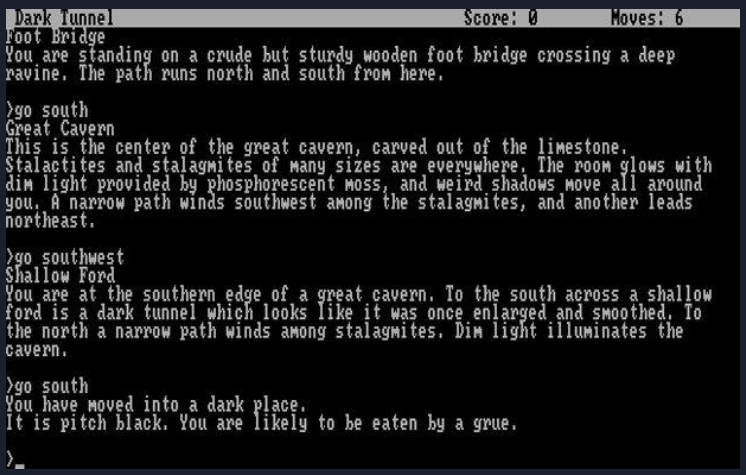


Deep Zork is a project focused on training an AI to play text-based games.

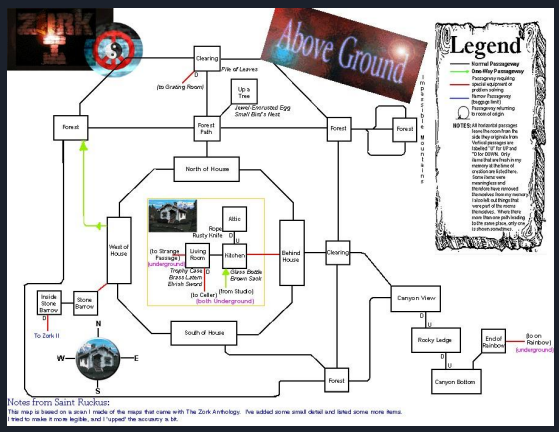
Text-Based Game AI Challenges

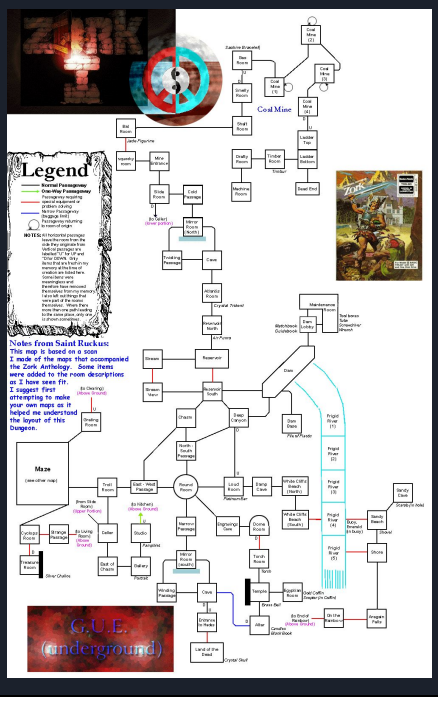
- Complexity
- Language comprehension
- Command select
- No visual information



While AI have been successful in beating games like Chess and Go and even Atari games, learning to play a text-based game presents a unique challenge. Instead of having visual information or all possible moves in the game readily available, to play a text-based game an AI has to be able to understand natural language to determine the most advantageous move at any given time.

Zork: The Great Underground Empire





In Zork the main goal is collect all 19 treasures found across the game world. Here are two maps of the game world - the above ground and below ground. Note that the player always starts at the West of House area on the above ground map.

In order to get the game to interface with Python, the subprocess module was used to connect a Windows executable, **Frotz** (https://github.com/DavidGriffith/frotz), which can run the game file. This allowed for both read and write access to the game from the Python environment.

Commands

1) Basic commands

- Go north, go south

- Look, check inventory

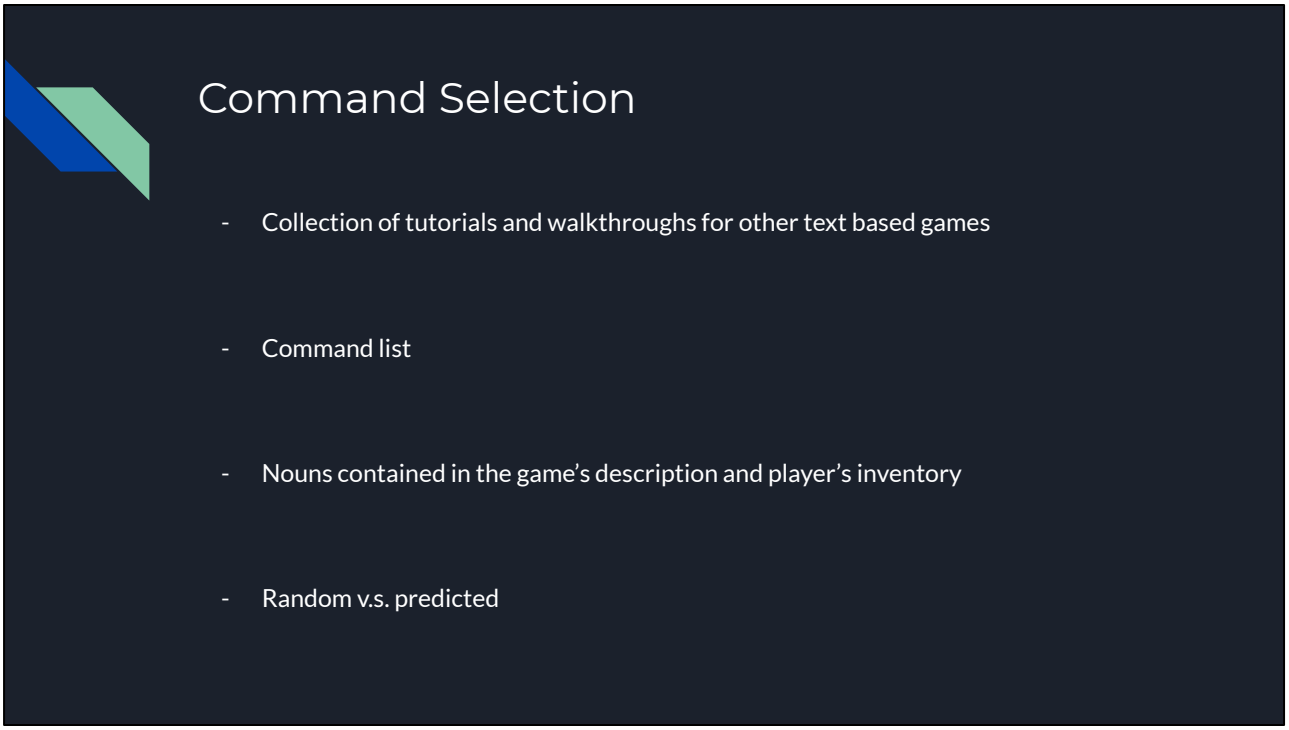
2) Verb-Object commands -

- Take key, open the door

3) Verb-Object-Prep-Object commands - commands that consist of a verb and noun phrase followed by a preposition and second noun phrase

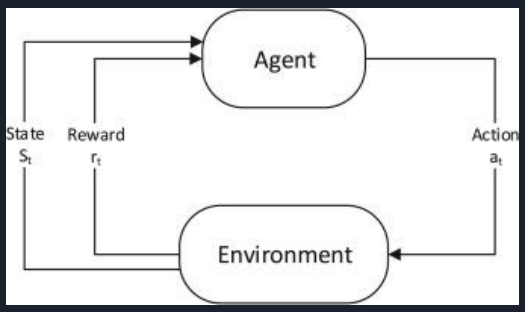
- Attack the monster with the sword, unlock the chest with the key

There are three types of commands for a game like Zork. Basic commands which include directional commands, verb-object commands that consist of a verb and a noun or noun phrase, and verb-object-prep-object commands that consist of a verb and noun phrase followed by a preposition and second noun phrase.



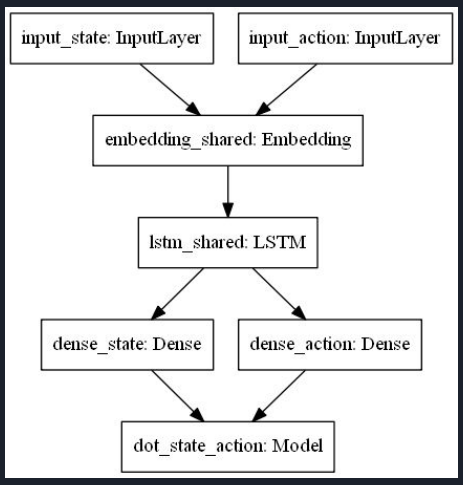
To select commands likely to be valid and advantageous, three methods were implemented. First, a corpus of tutorials and walkthroughs of other text based games was compiled. This allowed for testing possible commands to see if they are similar to the commands found in tutorials. Next, a command list for all possible commands used by Zork specifically was compiled by examining the game files. Then, by grabbing all the nouns in the game's description and player's inventory, all possible combinations of commands are generated and then scored using the corpus of tutorials and given a weight. Finally, it is decided whether or not to use a random action or predict which action will be best. This is chosen by a decaying factor known as epsilon which decreases each turn and leads to more and more predicted actions as time goes on.

Reinforcement learning



This a diagram of how reinforcement learning works at a basic level. There is an environment, or the game, and an agent which is the AI playing the game. The AI selects an action and performs it. The environment is then fed the action and the result of that action is sent back to the agent as the state of the game as well as a reward awarded for performing that action.

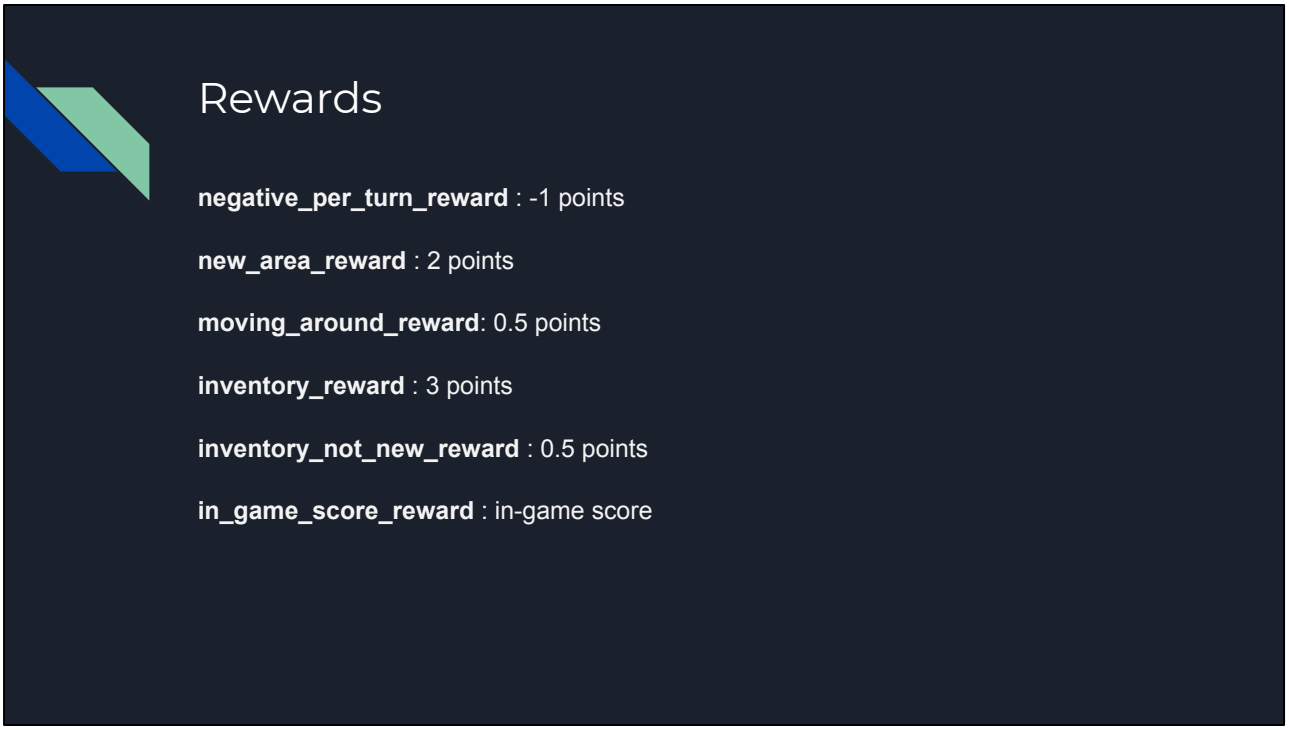
Double Deep-Q Network



$$Q(s_t, a_t) = r_t + \gamma \cdot maxQ(s_{t+1}, a_{t+1})$$

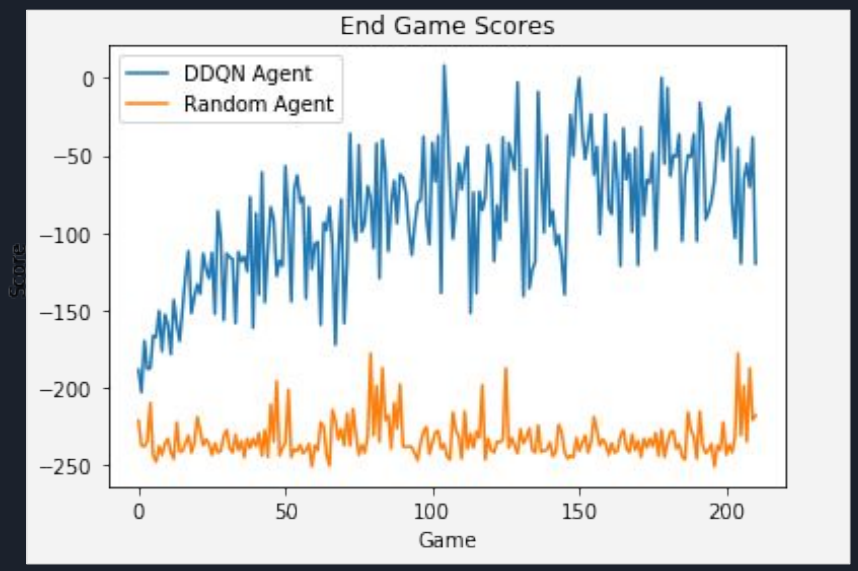
On the left is the neural network model used for this project. Both the state and the action are fed in separately to an embedding layer that captures higher dimensional data from the two. Then the data is passed into a long short term memory layer which works to capture the meaning of the words in the states and actions and takes into account the order of the phrases passed in. The state and action are then separated out to dense layers of lower dimensions (8 in this project) and then finally combined into a numerical value by taking the dot product of the two.

What's known as a double deep-q network or DDQN was used to train this AI. This type of model has been shown to be moderately successful in reinforcement learning applications. The network is based on the Q function seen at the right. Basically, the Q-value for an action in a given state is equal to the reward given for performing that action plus the maximum reward of all possible actions in the next state following the action. This future reward is then discounted by a discount rate known as gamma (0.75). Since the Q value of the next state is a prediction, it is important to discount it and the actual reward may end up being less.
The network is a double network because there are actually two models - one used for prediction of Q values and the other that is trained upon. Ever so often, the weights of the trained model are transferred to the prediction model. This helps stabilize the learning process otherwise the Q values can explode and the model can spiral out of control.



Each turn taken the negative_per_turn_reward will be added to the total reward for the round. This way, on turns where no other points were scored, the turn will be remembered as being non-productive. Discovering a new area or room in the game should be rewarded in order to encourage exploration. In order to prevent the agent from staying in one room too long, a small positive reward is given each time the agent moves from one area to another that has already been visited. This also helps to encourage the agent to return to an area it previously left when it had not yet completed all objectives in that area. Providing the moving_around_reward instead of the new_area_reward when the agent revisits an area will prevent it from exploiting the reward system and hopping back and forth between rooms. If the agent picks up an item or uses an item, the in-game inventory will change. This will be rewarded with a larger number of points as finding the treasures of the game ultimately is how the agent can win. In order to prevent the agent taking advantage of the large inventory_reward, a much smaller inventory_not_new_reward will be given when an inventory change occurs that has already taken place. This will prevent exploitable situations in which the agent repeatedly picks up and then drops an item. Scoring points in the game is the best type of action the agent could take so it should be rewarded the greatest. Each time an in-game score is granted, a weight (initially set at 10) will be applied to it to show its significance to the agent.

Experimental Results



After training for 210 games consisting of 256 turns each, the results of both the AI as well as a dummy model which only chosen random actions can be seen. As you can see, the AI greatly outperformed the random agent. However, even after over 200 games of training, the AI still fails to average over 0 total points in each game. It is important to keep in mind though, that the score referred to in this chart is not the in-game score but instead the rewarded score.

Going forward, there is much that can be done to improve the performance of Deep Zork. Below is a list of all planned future work and investigation. Training on the game data was the most computationally expensive part of this project. Calculating the max possible Q value for each state involves a large amount of iterations. Finding a way to cut down the action space size could be one way to improve the performance. Using a better method of predicting which commands are likely to be successful could speed up training by a large factor. Another option would be to parallelize the training and separate it completely from playing the game. A large amount of game data could be collected and then all the training could take place across multiple threads or machines. Because the state consists of the surroundings and the inventory and there are many items in the game, many combinations of surroundings and inventories can occur resulting in a very large state size. Currently the game stores all of these as separate states which leads to a rather large number of states. Finding a way to combine similar states could reduce overall state space size. Another option would be to feed the inventory and surroundings into the agent separately and create separate pipes for them within the model.

The reward values and conditions chosen were done so somewhat arbitrarily. Different values for the rewards could be tested as well as removing or adding additional conditions. The logic behind adding rewards for exploration and item interaction was to help the AI more quickly learn what it needed to do to beat the game. Although it might involve a longer training period, using only the in-game rewards could result in a better performance in the long run. Using a DDQN might not be the optimal form of reinforcement learning for the task of playing text-based games. More experimental networks could be tested and compared to the performance of the DDQN. An ensemble of networks could also be tested and might provide a more robust solution. Zork was chosen because it is one of the most well-known text-based games. However, it may not be the most friendly to reinforcement learning compared to other games. Finding more, simpler interactive fiction games with smaller game maps could be tested. Creating an agent that can successfully play a simpler game could then be translated to more complex games.