Jena Bada, James Kim, Sojung Yun
CS 175: Project in AI
March 24, 2023

Project Final Report - Minecraft: Tranquil AI

Project Summary

A lot of games have AI controlled bots acting as players to play the game. For example, a player fights against a bot in Super Smash Bros. In this case, the player will not attack the bot on a fixed path or direction. Accordingly, the bot must choose the action based on the player's actions. In that case, we need to gather data about all possible scenarios in which the player attacks so that the bot can figure out what action to take by analyzing all the data. This is actually done by artificial intelligence systems. The Tranquil AI that we have created also uses AI for this purpose. Tranquil AI is designed specifically to avoid combat with mobs in enclosed environments. The goal is to have the agent use different strategies to stay away from neutral and hostile mobs. The agent is placed in an environment where there is one opposing mob. By moving forward, backward, left, and right, the agent can avoid the mob. Our AI aims to ensure that the agent can defend against all mobs in an enclosed setting within a set timeframe.

The enclosed environment has a size of 32 x 32 and a time limit of 1 minute. The enclosed environment is critical as in an open environment, the agent can simply move in one direction to achieve an optimal result given that the agent is faster or as fast as the enemy mob. The agent is capable of moving forward, backward, left, and right.We chose Nearest-Neighbor Q-Learning to complete the objective. The Nearest-Neighbor Q-Learning method uses the K-Nearest Neighbors method to estimate Q-values. This is preferable compared to the general Q-Learning which uses a table to store action values, since storing all possible state-action pairs in a table would take too long to obtain the Q-values. In addition, K-Nearest Neighbors in Nearest-Neighbor Q-Learning is a non-parametric model which makes no assumptions based on the model underlying the process of data generation. Therefore, this method fits flexible models, and training the model can be faster than saving the data because it does not require parameter estimation. We could also use Deep-Q learning, but Docker image does not support most deep learning libraries Thus, Nearest-Neighbor Q-Learning is the best solution.

Approaches

We started this project out with a wide set of actions for the agent to carry out and decided to create an enclosed environment with a simple Q-Learning model with states that consisted of a square grid of block observations local to the agent, agent location, and nearest enemy location. The agent would also be rewarded for time survived and punished for dying. The coordinates

were regrettably not rounded. The possible actions had included more possible movements such as jump (in a direction) and attack.

Our next approach was to change the coordinates for the states to be rounded, the environment to be made smaller, and the number of possible actions to be reduced to just 4. The goal was to be able to achieve survival quite quickly with a more limited environment and the number of possible states were reduced many folds.

After analyzing the weaknesses of our initial approach, we decided to set survival as our baseline and decided to proceed with utilizing a Nearest Neighbors Q-learning algorithm. The Docker image doesn't support many deep learning libraries such as TensorFlow, so using the K-Nearest Neighbors classifier from the Sci-Kit Learn library was our next best choice. Nearest neighbors Q-learning is a variation of the Q-learning algorithm used in reinforcement learning. In nearest neighbors Q-learning, instead of using a table to store the action-values, the algorithm uses a k-nearest neighbors approach to estimate the Q-values for each state-action pair. This means that instead of using the exact Q-value for a state-action pair, the algorithm estimates the value based on the average of the Q-values of the k-nearest state-action pairs in the training set.

For our AI, we defined a state as a tuple of the agent's x-coordinate, y-coordinate, and distance from the mob. Unlike the traditional Q-learning model where you store one Q-value per state, we stored a tuple of 4 Q-values, each of which were associated with the up, down, left, right action the agent can take.

For reward, the agent would gain more reward for having a greater distance from the mob.

To train the model the Nearest Neighbors Q-Learning, we used the following algorithm described below:

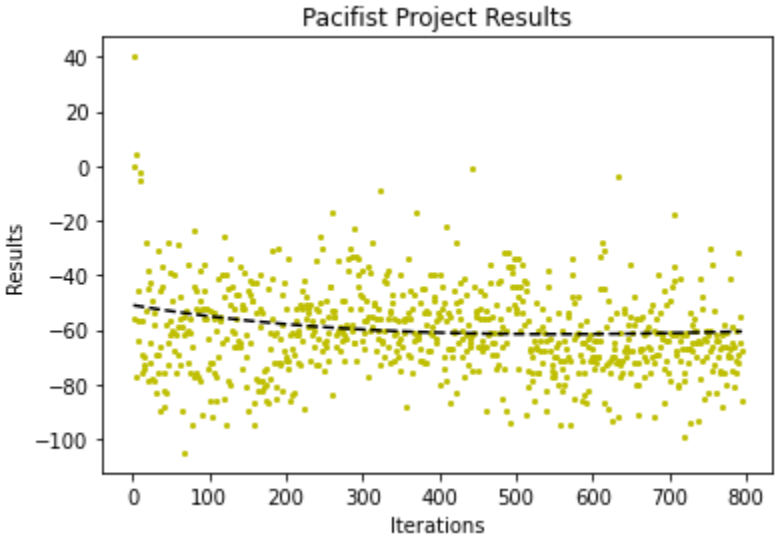
1. Initialize kNN model.
2. Set-up a for-loop to run for 100 epochs.
3. Within each epoch, do the following:
    a. Get the current state of the agent.
    b. When deciding on the agent's next action, either choose to do a random one or have the model predict one using kNN depending on the epsilon value.
    c. Take action, and observe the new state and reward.
    d. Calculate new Q-values and store state and action tuples.

To combat our problem in our initial approach of not being able to store every possible state, we implemented experience replay. Instead of storing every game state that our agent encountered, we would store a maximum of 1000 states in our experience replay memory. Once that was filled, we'd take a random subset of the data, calculate new action tuples for each one, and

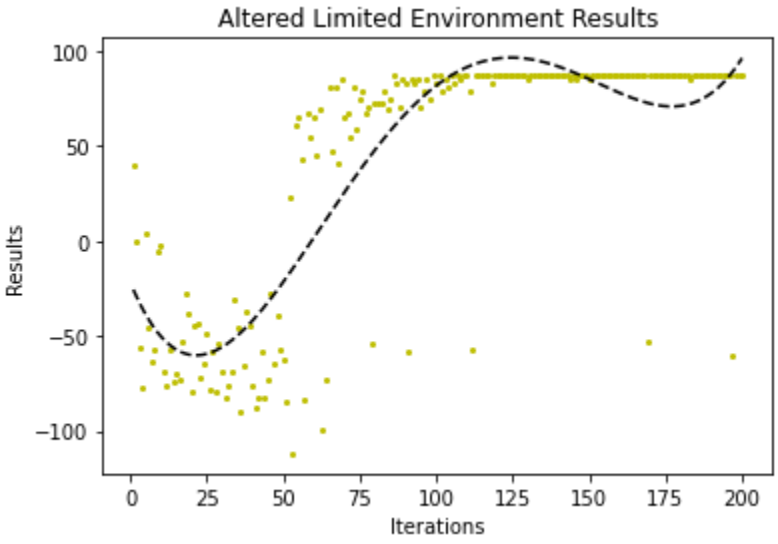
retrain the model using these values as a minibatch. For any subsequent epoch, we would overwrite old values in our memory with new state-actions.

Evaluation

For the Q-Learning prototype with a wide set of possible actions and unrounded agent/mob coordinates, it resulted in much higher variability which is why the result seems to look like a random scatter plot. There were too many possible states and actions that a simple model could not train effectively.



In the next approach we greatly limited the number of possible states and actions. With this, we achieved a very small goal of exhibiting a functioning model. However, this extremely limited environment led to a very specialized result and overfitting and couldn't be relied on to have similar results in a bigger environment due to the complexity. However, it was effective in its results which showed that the design was on the right track.

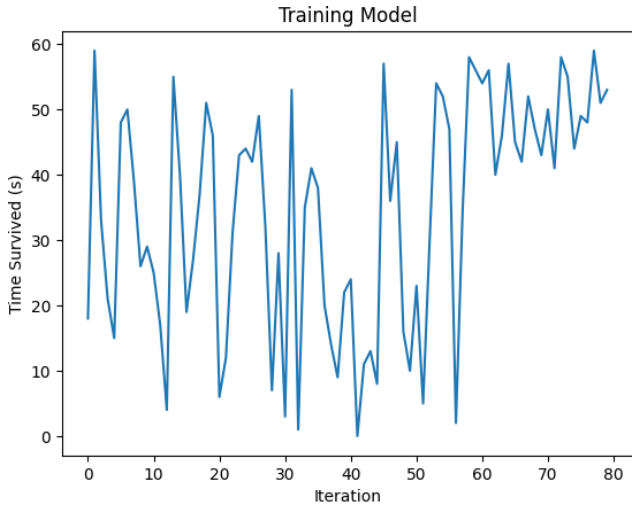


To quantitatively evaluate the performance of our AI, we tuned the hyperparameters of the kNN model. The main hyperparameter that could be changed in a kNN model is the number of neighbors (k). We decided to not train the model with a small k such as 1 since we wanted the model to consider more states before making a decision. When training the model with k=10, we found that the agent was not performing well when it encountered new states. This meant that the model was overfitting the data. Additionally, since increasing k meant that there was an increase in the computational resources needed to make a prediction, this made making a prediction on

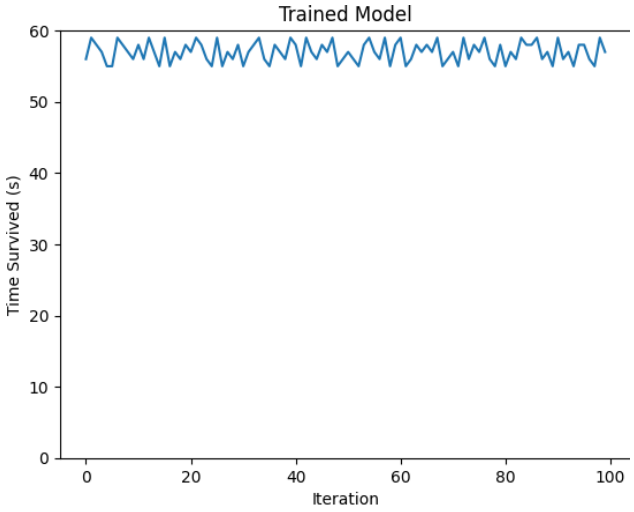
the model slower thus slowing down our agent's decision making process. Because of this, we decided to decrease our k to 5, which yielded optimal results.

To qualitatively evaluate our agent, we compared how long the agent survives using an untrained model against a fully trained one.

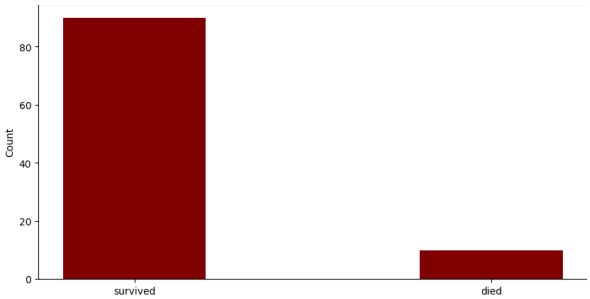
The training log of our KNN Q-Learning model showed success in the new implementation in the enlarged environment (physical environment). With this model, our evaluation of the success of the model was based on the survival time of the agent. The model exhibited a gradual improvement in performance and survival local minimum times as the number of training went up.

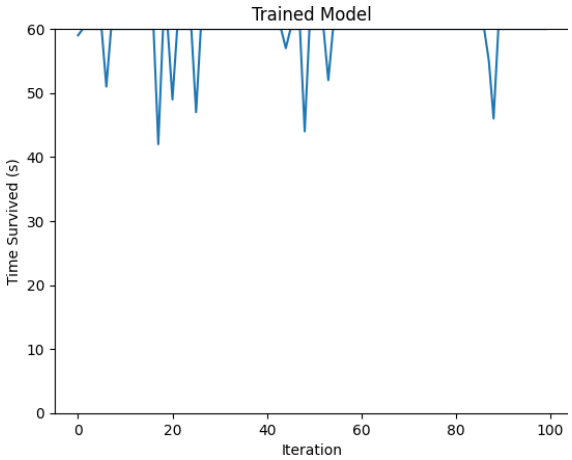


Using the fully trained model, the results showed success with a fairly consistent survival time. The trained model showed better consistency than the high iteration results of the training model because the trained model used a lower non-zero epsilon in order to avoid getting stuck in local optima and preserving reliability.



The trained model showed an X% improvement over the training model.





References and Resources Used

Shah, Devavrat, and Qiaomin Xie. "Q-learning with nearest neighbors." Advances in Neural Information Processing Systems 31 (2018).
https://github.com/microsoft/malmo/tree/master/Malmo/samples/Python_examples
https://microsoft.github.io/malmo/0.21.0/Schemas/MissionHandlers.html
https://microsoft.github.io/malmo/0.30.0/Schemas/Types.html
https://towardsdatascience.com/k-nearest-neighbors-k-nn-explained-8959f97a8632
CS175 Assignment 2 python file