24CSAI04H

AI Planning for Robot Systems

Phase 2 Report

A Robotics Simulation of Krusty Krabs Restaurant

ID	Name	Email
224030	Amir Mohamed	amir224030@bue.edu.eg
226392	Merihan Ahmed	merihan226392@bue.edu.eg
228248	Omar Emara	omar228248@bue.edu.eg
218767	Sameh Ayman	sameh218767@bue.edu.eg

Table of Contents

Introduction

Once the Krusty Krab closes at night, Mr. Krabs attempts to painstakingly collect money that a rich customer accidently forgot behind. With his "sniffing" senses, Mr. Krabs' robot systematically scans the restaurant for any sign of that money that might have been left behind. The closer he gets, the stronger the "sniffing" reward becomes, thus leading him to his target. He spends the whole night tracking down every corner of the Krusty Krab until he reaches it. To achieve this, we use Double Deep Q-Network (DDQN) to allow him to “sniff” the money and determine the best direction to take.

Objectives

- Implement a DDQN algorithm to train a single agent to navigate his environment and reach the specified target efficiently.
- Successfully train the agent to locate and reach the intended target within the environment using exploration and exploitation strategies.
- Define a proper reward system suitable enough to nature of the environment that can incentivize the agent to find the money in an efficient and timely while avoiding static obstacles all around.

Overview

The project involves designing a simulation where an agent, Mr. Crabs, utilizing a Double Deep Q-Network will navigate himself toward finding and reaching the target-money in a continuous environment that includes obstacles and must learn the most efficient path to the target. In due process, the agent, through reinforcement learning, will make use of the feedback provided as reward or otherwise to shape its behavior in order to optimize its decisions. In essence, the agent learns to navigate its environment by exploring and exploiting it, with the aim of eventually reaching the target-money-while avoiding static and dynamic obstacles. It will learn an improved navigation strategy over time through interactions with the environment and receiving a reward for each of its actions.

Environment Design

Both the environment and assets used were specifically designed to a *lore-accurate* replica of the Krusty Krabs restaurant from the original show. Everything except Mr.Krabs was created using Ibis Paint X.

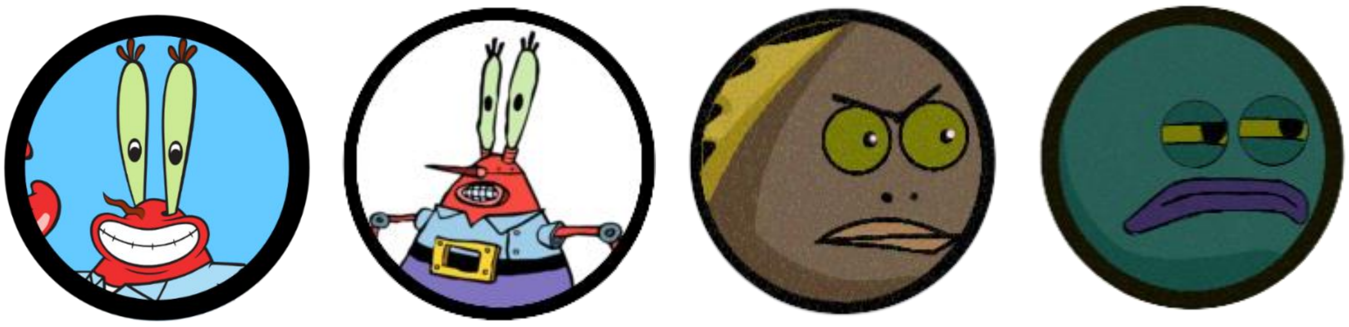


Figure 1: Mr.Krabs and angry customers (dynamic obstacles)

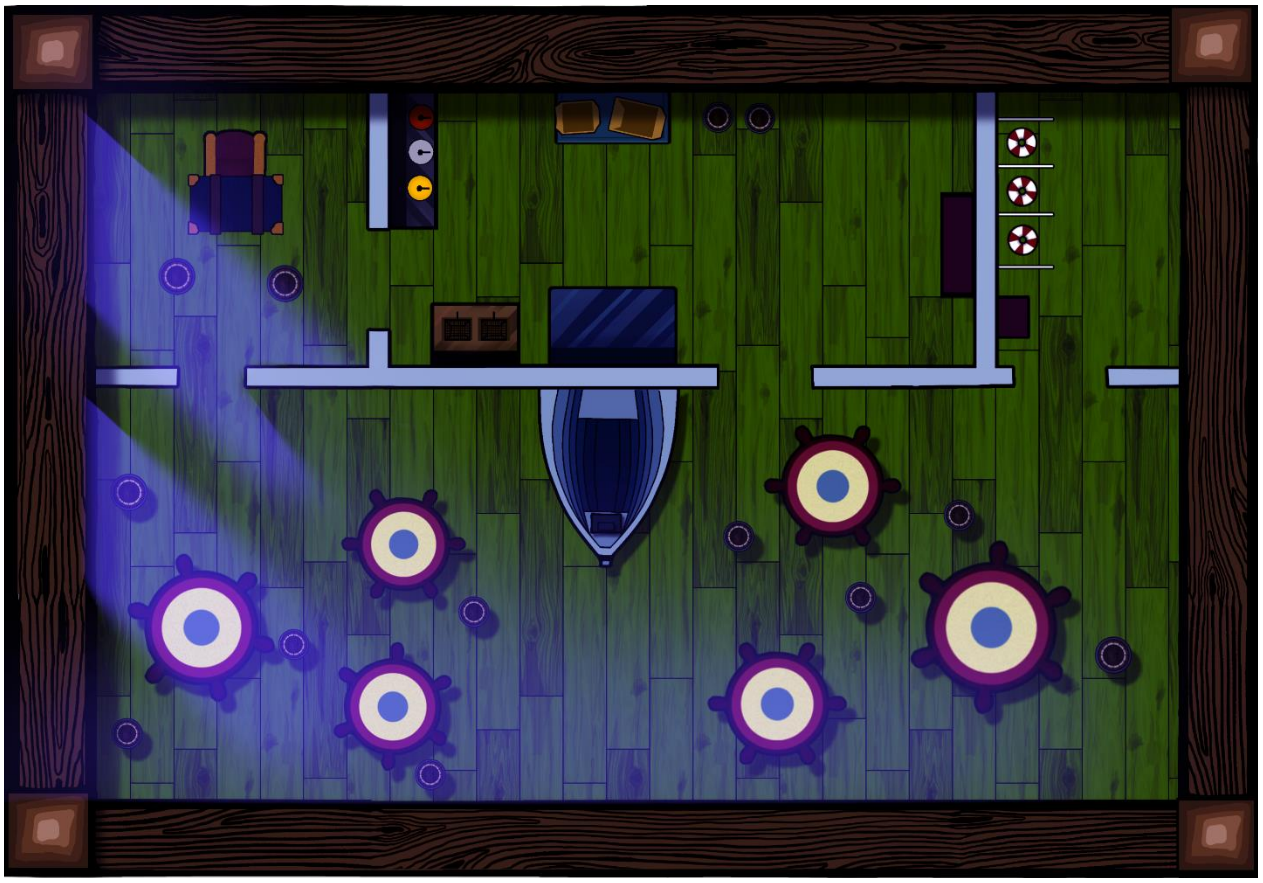


Figure 2: The Krusty Krabs main environment (At night)

Design Decision

1. Reinforcement Learning Algorithm Overview

Initially, we used Deep-Q-Network (DQN) and its simpler counterpart QN before it as the algorithms of choice. However, due to the inherent complexity of our environment, being a 238 by 165 grid, in addition to the obstacles as well as training problems we faced, we ended up utilizing Double Deep Q-Network (DDQN). DDQN is an improvement over the standard DQN algorithm in that it is designed to prevent the overestimation of Q-values by breaking the relationship between the action selection and the value estimation resulting in an improved overall stability and accuracy. It uses two networks: a policy network, which is used for the selection of actions during both training and inference, and a target network, which provides stable Q-value targets for training; this target network is updated at a periodic interval to avoid instability. Instead of directly using maximum Q-value from the target network, Double DQN utilizes the policy network for action selection and then evaluates those selected actions by the target network to properly avoid overestimation errors. It stores historical experiences in tuples so that the agent can sample from them in order to break correlations found in sequential data, thus making learning more efficient. Epsilon-greedy exploration strikes a balance between exploring new actions and exploiting optimal actions by gradually reducing the likelihood of taking a random action over time. The loss function, which is defined as the mean squared error between the target and predicted Q-values, ensures the policy network will be trained to approximate the optimal action-value function.

2. Neural Network Architecture

A feedforward network with three hidden layers, each containing 64 neurons and ReLU activation. This design provides sufficient capacity for learning complex state-action mappings without being overly complex.

3. Hyperparameter-tuning

Target Network Update Frequency:

- We made it so that the target network updates every 100 episodes. This allows it to balance between providing stable training targets and keeping the policy network aligned with the target network.

Replay Memory Size:

- A huge memory size of 100,000 was chosen to prevent catastrophic loss / policy degradation with time and ensure a more stable training process.

Batch Size:

- A Batch size was selected to be 32 to capture as much useful information as possible.

Learning Rate:

- We experimented with a variety of learning rates and concluded that the best option is 0.0001 because it reduces the risk of large Q-values oscillations, giving the model time to explore and learn the best course of action.

Exploration Strategy:

- The epsilon-greedy approach gradually shifts from exploration (EPS_START = 1.0) to exploitation (EPS_END = 0.01) over the course of training with a decay rate (EPS_DECAY = 0.998).

4. Reward

Our reward system is comprised of various moving parts after experimenting and observing the training for our agents. It includes the following:

- Most important of course is reaching the target itself as it is the primary goal after all. The agent receives a reward of 50.0 when it successfully reaches the target area.
- Additionally, the agent is awarded the equivalent of the Euclidean distance between its current position and the target. This only occurs when the distance is smaller compared to the distance from the previous step.

5. Penalty

- On the other side, if the distance is larger than the previous step, the agent is penalized by the Euclidean distance -0.15 to discourage inefficient and exploratory moves that may drive it away from the goal.
- Additionally, with each passing step, the agent is penalized by -0.01 to discourage excessive movement without meaningful progress.
- The highest penalty by far is when the agent makes an invalid move (trying to move into an obstacle). This immediately results in a penalty of -5.0 and the episode completely ends to avoid any unnecessary encouragement further.

Implementation Details

To implement the project, we have used a wide range of development and design tools. For the design part we used Ibis paint X to design the assets that later were used for the app. For development, we ended up using VS Code and Jupyter notebooks.

The app's code was written in Python and PyGame which is a popular python library for developing games. Many internal python modules such as math, random and time were incorporated as well. The code was written in a modular fashion making extensive use of best coding practices and was beautifully broken down into manageable functions and classes.
On a software level, we also employed NumPy for managing the occupancy grid and general matrix manipulations, as well as TensorFlow for RL policy design and OpenAI gym for the environment.

Challenges and Solutions

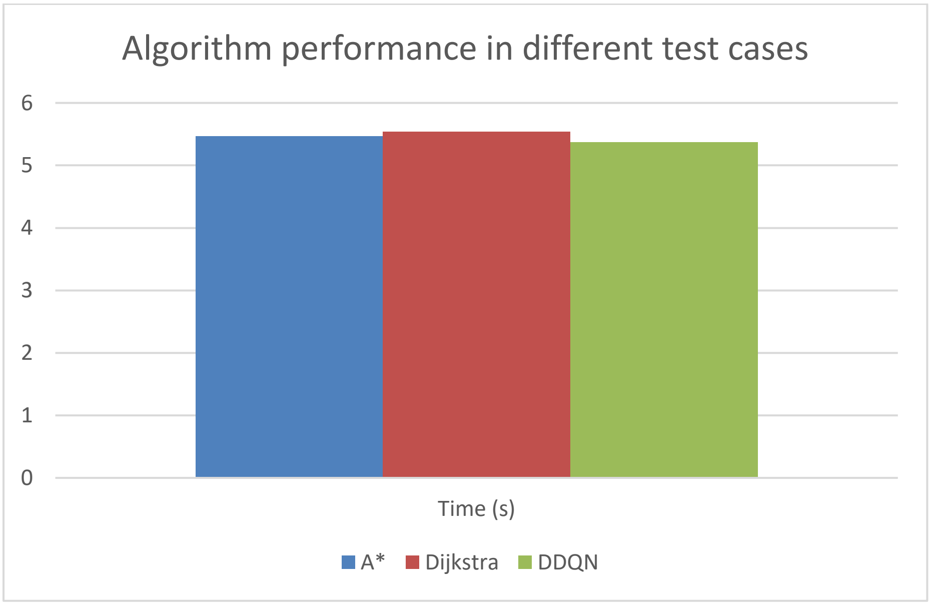
Implementing this simulation was indeed a tough task, we have gone through many challenges that we were not anticipating while planning the development process (considering we estimated that this will end in 1 day). The following are some of the “interesting” hurdles we faced:

Challenge	Solution
Complex Environment	We increased epsilon start to encourage further exploration initially. Additionally, we made a robust reward system that fits the needs of our environment.
Unstable Converging	Increased the overall complexity of the neural network by making it 3 layers of 64 neurons each.
Overfitting	Implemented DDQN instead of vanilla DQN to utilize the inherent mechanism that prevents overfitting.

Comparison and Results

We compare the algorithms based on time elapsed (in seconds) as a measure of efficiency. To equal the playing fields, there were neither dynamic obstacles nor mapping to properly capture their full potential in the ideal scenario. Additionally, all algorithms were given the same starting points as well as the same end point to reach one target. The results are shown below:

	Time elapsed (s)
A*	5.47
Dijkstra	5.54
DDQN	5.37



The results show that a properly trained DDQN agent can navigate our environment slightly better and faster than traditional search algorithms such as A* and Dijkstra. However, one thing to keep in mind is the complexity and overall time it takes to train said model in the first place. Additionally, the circumstances would skew the results dynamic obstacles were introduced in different scenarios.

References

[1] GeeksforGeeks, “what is difference between the DDQN and DQN?,” *GeeksforGeeks*, Feb. 10, 2024. https://www.geeksforgeeks.org/what-is-difference-between-the-ddqn-and-dqn/

[2] P. Bajaj, “Reinforcement learning - GeeksforGeeks,” *GeeksforGeeks*, Apr. 25, 2018. https://www.geeksforgeeks.org/what-is-reinforcement-learning/

[3] GeeksforGeeks, “How to Make a Reward Function in Reinforcement Learning?,” *GeeksforGeeks*, Oct. 2024. https://www.geeksforgeeks.org/how-to-make-a-reward-function-in-reinforcement-learning/ (accessed Nov. 25, 2024).