# CS238 Final Project
# Ocean cleanup using sea roombas

**Johanna Eriksson, Shreeya Khadka, Sicheng Zeng**


## Abstract

Plastic trash is an environmental nuisance plaguing the world's oceans. We explore how an imagined sea surface roomba might be used to collect plastic trash. Such innovative solutions are urgently needed, particularly since the most promising ocean cleanup technology failed last year after only a few months[1]. We use Q-learning to produce optimal policies for navigating the sea roomba from its starting point to a designated ending point, while picking up as many pieces of trash as possible along the way.


## Introduction

The issue of oceanic pollution is a challenging one with no obvious solution. Large companies and nonprofits such as The Ocean Cleanup[2], 4ocean[3] and the Surfrider Foundation[4] have been working on ocean cleanup for years without satisfying progress. There have been countless hiccups, with products failing and accidentally releasing collected trash back into the ocean. Even when functioning correctly, the existing solutions simply do not have enough impact to overwhelm the constantly growing amounts of pollution[5]. One reason for this is that most methods, such as The Ocean Cleanup's garbage-harvesting ships, require entire crews and weeks or months to slowly sail through polluted areas and conduct time-consuming cleanup.

We envision an automated cleanup method that could eventually be expanded to cover large stretches of the ocean surface. Autonomous robots—we call them sea roombas—would leave their station, collect trash and return, then rinse and repeat, requiring no human labor and minimal oversight. In this paper we use Q-learning to train policies that would guide our sea roomba to trash in its visible range and then return safely to its station. With these policies, we can be confident that the sea roomba can run automatically for long stretches of time, charging itself as needed and consistently removing trash from the environment.



[1]https://www.theverge.com/2019/1/9/18175940/ocean-cleanup-breaks-plastic-pollution-silicon-valley-boyan-slat-wilson

[2]https://theoceancleanup.com/oceans/

[3]https://www.4ocean.com/

[4]https://www.surfrider.org/

[5]https://www.theverge.com/2019/1/9/18175940/ocean-cleanup-breaks-plastic-pollution-silicon-valley-boyan-slat-wilson

## Related Work

### Navigation and path planning using reinforcement learning for a Roomba robot[6]

In this paper, the authors use reinforcement learning methods to find the best route from one location to another, such that a roomba can cover as much area as possible. The learning process is done with knowledge of the building floor plans, with the intention that the roomba would regularly clean the floors in a university, but that the methods could be similarly applied to homes, hospitals, and offices. While the goal of our sea roomba is not to sweep the entire area, but instead to pick up trash at observed locations, we can still use similar learning techniques.

### Constraint-based multi-robot path planning[7]

This paper addresses constraint-based multi-robot path planning. The author demonstrates that representing the problem of planning collision-free paths for multiple robots is a constraint satisfaction problem. He uses constraint propagation and intelligent search ordering which allows the size of the search space to be reduced significantly, compared to simpler techniques like forward search which goes through all possible actions. In our scenario, we envision eventually having multiple sea roombas navigating in the same space, which would make this work relevant.

### Vision-based reinforcement learning using approximate policy iteration[8]

Here the authors use the Least-Squares Policy Iteration (LSPI) reinforcement learning algorithm on a roomba for learning a docking task using vision as the only form of sensing. They show that LSPI and a new algorithm named LSPI+ learn the task quicker than other reinforcement learning methods that are used for comparison. While our project

[6]https://ieeexplore.ieee.org/document/7955160

[7]https://ieeexplore.ieee.org/document/5509582

[8]https://ieeexplore.ieee.org/document/5174696

focuses on Q-learning, the LSPI method might be a relevant algorithm to try in future work.

# Problem Statement

## Sources of uncertainty

The problem of sea trash collection has many sources of uncertainty. Even the best robots fail at times. The sea roomba may not succeed at picking up the trash it attempts to retrieve. The trash could snag on a piece of seaweed rooted to the ocean floor, or there could be aquatic life interfering, resulting in an action failing and needing to be repeated or abandoned.

There is also the factor of the ocean currents. Whatever basic current flows we simulate may never match the real thing, and thus nor will the roomba's movements in the water. Additionally, the roomba's sensors could fail and thus the estimated locations of other roombas or the end location may drift or otherwise become incorrect. In fact, the uncertainty of the roomba's own location may be one of its main challenges, as the water constantly shifts around.

And finally, in the water, the roomba is interacting with many outside agents - sea creatures, boats or swimmers, and potentially other roombas. These agents may bump into it or otherwise disrupt the roomba's path.

## Environment

In our project we simplify the environment to be a static 2D grid. We ignore state uncertainty, model uncertainty and interaction uncertainty. We keep some action uncertainty by allowing the roomba to only make successful actions for some percentage of the time. Since the roomba is in water, we assume this probability is linked to the water current which impacts the roomba's ability to take its desired action. In contrast, we assume that the plastic trash itself is stationary in the water and not affected by the current.

Formally, the environment is a $5 \times 5$ grid. We assume that at max $25\%$ of the grid is filled with plastic trash, i.e. we can have at max 6 pieces of trash. We also assume that the roomba knows the shape of the grid, its own position, its ending point, as well as the number of pieces of trash and their locations. We require the roomba to reach its ending point within $100$ time steps, representing its fixed battery life.

## Modeling the problem

The problem can be modeled as a Markov Decision Process (MDP) with the following components:

- **State space** $\mathcal{S} = \{(r_i, r_j), (t_{1i}, t_{1j}), (t_{2i}, t_{2j}), ...\}$ where $(r_i, r_j)$ denotes the position of the roomba and $(t_{xi}, t_{xj})$ denotes the position of the $x$th piece of trash, for up to $5 \times 5 - 2 = 23$ pieces of trash. Trash is assumed to not exist in the robot's starting or ending position.

  Therefore, we denote:

$$R = \{\text{valid positions of roomba}\}$$

$$T = \{\text{all possible trash allocations}\}$$

  The total number of states in our problem can be calculated by:

$$|R| \times |T| = 25 \times 2^{23} = 209,715,200$$

- **Action space** $\mathcal{A} = \{0 : \text{left}, 1 : \text{right}, 2 : \text{up}, 3 : \text{down}\}$ which lets the roomba move one step in the grid in either direction.
- **Observation space** $\mathcal{O} = \{\text{wall, water, trash, start, end}\}$ where "wall" indicates whether the roomba is in contact with a wall, "trash" indicates whether the roomba has reached a trash location, "start" and "end" indicate if the roomba has reached its starting and ending positions, and "water" represents all other positions in the grid.
- **Reward model** $R(s, a)$ is:
  - Pick up trash: $+10$
  - Wall collision: $-5$
  - Take a step: $-0.1$
  - Timeout: $0$
- **Transition model:** We test two transition models $T(s, a, s')$. The first is deterministic in the sense that the roomba always succeeds in stepping in its desired direction, unless it collides with a wall. The second is probabilistic in the sense that the roomba might not be able to take its desired step and ends back in its old position, i.e. $s = s'$. The objective is to simulate uncertainty due to water currents impacting the roomba's ability to move in its desired direction.
- **Discount factor** $\gamma$ is set to $0.95$.

The roomba can navigate the grid for $N = 100$ time steps. We imagine these time steps as representing the roomba's battery life. The ending point is a terminal state. The roomba needs to balance going after pieces of trash with reaching its ending point on time, as shown in Fig. 1.

# Methods

## Baseline

Our baseline method is a random policy where, for each time step, the roomba randomly chooses a direction to go in. It stops when hitting the maximum $N = 100$ time steps or reaching its terminal ending point.

## Reinforcement Learning

We use the Q-learning algorithm with eligibility traces to estimate the optimal policy for the roomba. We incrementally update the Q action value function using the following update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s, a) + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

```
  0 1 2 3 4
 -------------
0| E . . t . |
1| t t R . . |
2| . . t . . |
3| t . . . t |
4| . . . . . |
 -------------
Took action 3 with reward -0.1

  0 1 2 3 4
 -------------
0| E . . t . |
1| t t . . . |
2| . . R . . |
3| t . . . t |
4| . . . . . |
 -------------
Took action 3 with reward 10.0
```

Figure 1: Roomba (R) moving around the grid with pieces of trash at locations (t) and ending point at (E).

where $\alpha = 0.75$ is the learning rate, $\gamma = 0.95$ is the discount factor, $R(s, a)$ is the reward obtained by taking action on $a$ at state $s$, $s'$ is the next state, and $a'$ are all possible actions that can be taken from $s'$.

For faster convergence of the algorithm, we use eligibility traces. In each episode of the roomba's trash collection, we first let the roomba navigate until termination and then back-propagate the rewards from its ending state to its starting state.

### Training Details

We train our algorithm on a $5 \times 5$ grid with $10,000$ randomly generated grids that have different numbers of trash counts and different probabilities of making successful moves. We assume that we know the upper limit of the trash count which is 6 or $25\%$ of the grid locations.

However, we do not know the strength of the water current, hence we randomly sample the probability of successfully taking an action from a uniform distribution between $(0.5, 1)$. A move probability of $0.8$ indicates that the water current is low enough for the roomba to successfully step in its desired direction $80\%$ of the time. Each training grid is run $100$ times to stabilize the Q-values update per grid. We follow two training schemes as listed below:

- **TrainCaseA**: The roomba always successfully performs its desired action.
- **TrainCaseB**: The roomba's probability of successfully taking an action is sampled from a uniform distribution between $(0.5, 1)$.

## Results

### Testing Details

Similar to the training stage, we test our algorithm on a $5 \times 5$ grid with $100$ randomly generated grids that have different numbers of trash counts and different probabilities of making successful moves. We know the upper limit on the trash count which is 6 or $25\%$ of the grid locations. We test the following two cases:

- **TestCaseA**: The roomba always successfully performs its desired action. Training is performed on **TrainCaseA**.
- **TestCaseB**: The roomba's probability of successfully taking an action is sampled from a uniform distribution between $(0.7, 0.9)$. Training is performed on **TrainCaseB**.

Since each grid can have a different number of rewards, we scale the total reward at the end of an episode by the total number of trash in the grid. Hence the upper limit on our average reward collected is 10, which is the baseline reward given to the roomba according to our reward model. We then plot the evolution of performance of our algorithm over time for both **TestCaseA** and **TestCaseB**.

In Figures 2 and 3 we observe how both the average rewards and standard deviation of rewards start converging towards the end for **TestCaseA**, where we have a fixed transition probability of 1. This tells us that our model is able to learn a policy that is not only close to optimal but also generalizes well across multiple grids.

In Figures 4 and 5 we observe that while both the average rewards and standard deviation of rewards start converging towards the end for **TestCaseB**, where we have variable


Figure 2: Average scaled rewards collected by the roomba over time for **TestCaseA**

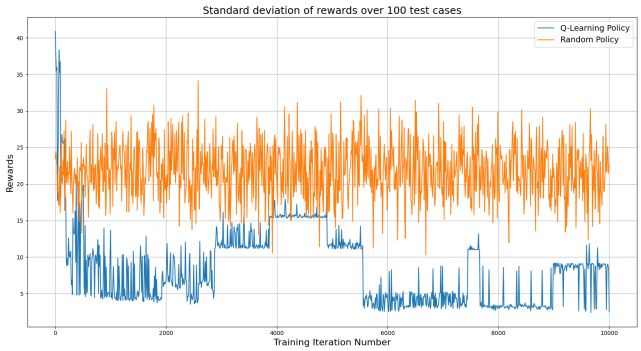


Figure 3: Standard deviation of scaled rewards collected by the roomba over time for **TestCaseA**

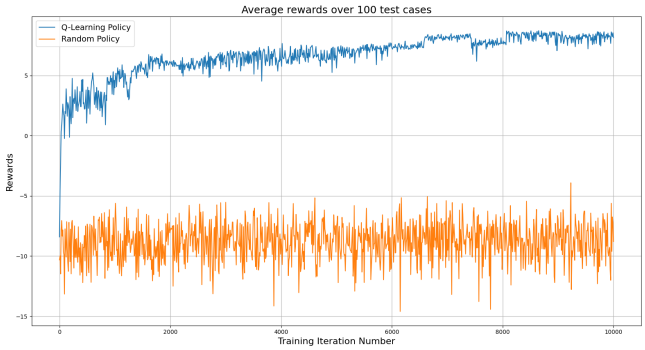


Figure 4: Average scaled rewards collected by the roomba over time for **TestCaseB**

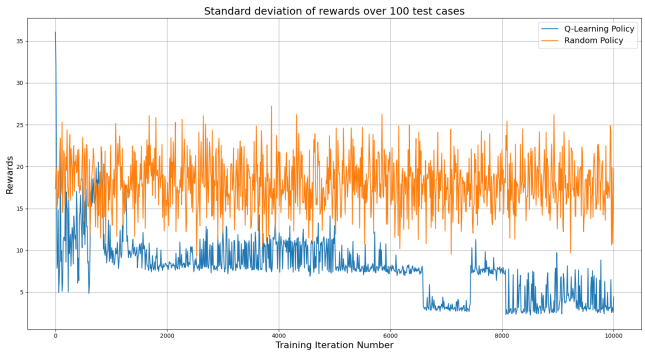


Figure 5: Standard deviation of scaled rewards collected by the roomba over time for **TestCaseB**

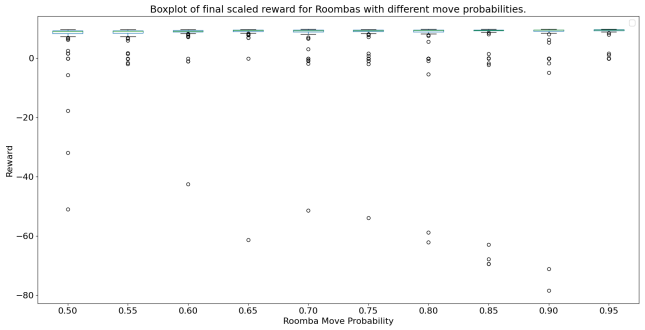


Figure 6: Boxplot of scaled rewards collected by the roomba with different move probabilities across 100 boards after $10,000$ iterations.

transition probabilities, the convergence is not as good as in **TestCaseA**. Towards the end we see that as the average reward increases after $8,000$ iterations, the standard deviation of rewards also increases. This tells us that although our model is able to learn a policy that is very close to optimal, it is not able to generalize well across multiple grids with different transition probabilities, i.e. across different water current levels. This indicates that we might need different models depending on the water current level itself.

### Summary of results

Table 1 shows our final results. Q-learning outperforms the random policy over time and generalizes across multiple test cases.

Table 1: Comparison of scaled rewards

| TestCase | Method | Mean | SD |
|---|---|---|---|
| TestCaseA | Baseline | -10.63 | 22.56 |
| TestCaseA | Q-learning | 8.84 | 2.51 |
| TestCaseB | Baseline | -8.78 | 19.75 |
| TestCaseB | Q-learning | 8.14 | 4.45 |

To observe how well our policy generalizes over different transition probabilities (i.e. different water current levels), we plot the performance for 100 grids, each with a fixed probability of the roomba successfully completing its desired action. This is shown in Figure 6. After $10,000$ iterations, the Q-learning algorithm converges to a stable policy across all move probabilities with few outliers. We also note that the interquartile range of scores decreases with increasing probability of success. This means that we have more stable performance for lower water current levels than for higher water current levels.

## Conclusions

We address the issue of oceanic trash by imagining autonomous sea roombas that can efficiently clean up the ocean without the need for big boats and crews. We use Q-learning to produce optimal policies for navigating the roomba from its starting point to a designated ending point, while picking up as many pieces of trash as possible along the way.

We model the environment as a 2D grid representing an area of the ocean. We require the roomba to reach its designated ending position within 100 time steps, as a representation of its fixed battery life. We assume that the roomba knows the shape of the grid, its own position, its ending point, as well as the number of pieces of trash and their locations. We model the problem as a Markov Decision Process (MDP).

Our baseline method is a random policy where, for each time step, the roomba randomly chooses a direction to go in. It stops when hitting the maximum $N = 100$ time

steps or reaching its terminal ending point. We use the Q-learning algorithm with eligibility traces to estimate the optimal policy for the roomba. We train our algorithm on a $5 \times 5$ grid with $10,000$ randomly generated grids that have different numbers of trash counts and different probabilities of making successful moves. We then test our algorithm on $100$ grids which similarly have different numbers of plastic trash counts and different probabilities of making successful moves.

We have two cases. In TrainCaseA and TestCaseA, the roomba always successfully performs its desired action. In TrainCaseB and TestCaseB, the roomba's probability of successfully taking its desired action is sampled from a uniform distribution. In TestCaseA, both the average rewards and standard deviation of rewards start converging towards the end. This tells us that our model is able to learn a policy that is not only close to optimal but also generalizes well across multiple grids. In TestCaseB, although both the average rewards and standard deviation of rewards start converging towards the end, the convergence is not as good as in TestCaseA. While our model is able to learn a policy that is very close to optimal, it is not able to generalize well across multiple grids with different transition probabilities, i.e. across different water current levels.

From our results we see that it is very feasible to train a model to guide a roomba to pick up trash in the ocean, at least under preliminary assumptions such as knowledge of the roomba's current location and location of the trash. These movement policies can be used to guide sea roombas to clean up the world's oceans in a completely automated way, with no need for human labor, in a scalable and sustainable fashion.

## Further Work

This project could be extended in several ways to more realistically simulate the complex ocean environment:

- The roomba's route should avoid collision with other sedentary or active sea organisms. We could do so by adding creatures in the board with negative rewards on collision.
- We could simulate the movement of trash and sea creatures more realistically by adding random move probabilities to both.
- In our simulations, we also noticed scenarios where the roomba would go back and forth between two positions and eventually reach timeout. We believe that an experience replay buffer would have been helpful to mitigate redundant updates in these scenarios. These scenarios represent some of the outliers in Figure 6.
- We could use neural networks to estimate the state-action function itself on the 2D grid, which would reduce the number of states we have in the simulation.

## Contributions

Three people contributed to this project: Johanna Eriksson, Shreeya Khadha, and Sicheng Zeng. The MDP simulation and visualization were created by Sicheng Zeng, the Q-learning algorithm and related figures were implemented by Shreeya Khadha, and a sparse sampling algorithm (we did not end up including it in the paper) and the majority of the writeup was worked on by Johanna Eriksson. Everyone contributed to the writeup in some way.

https://github.com/ClearSpear/cs238-final-project