

CS - 520

Introduction to

Artificial Intelligence

Project 3

By

Deviram Kondaveti (dk1273)

Vaishnavi Madhavaram (vm695)

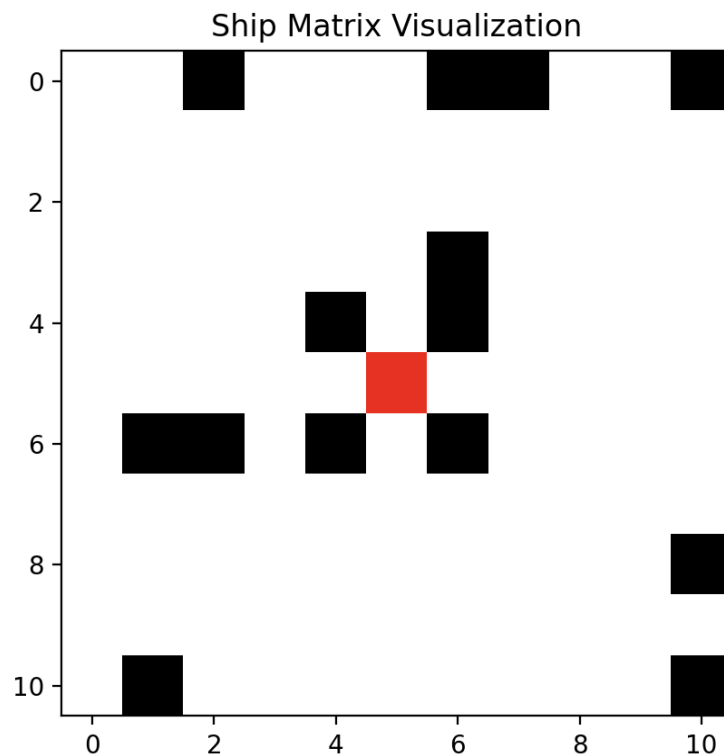
Puja Sridhar (ps1393)

PROBLEM STATEMENT:

In the aftermath of fighting an alien invasion, the bot must assist its crew, who has been affected by psychoactive alien spores. The crew is now in a terrified state, moving erratically throughout the ship. The objective of the bot is to navigate through the open cells in a ship of size 11 X 11 to strategically guide the crew member to the teleportation pad for immediate evacuation and medical attention.

SHIP GENERATION:

The ship for this project is an 11 X 11 grid. There are a total of 14 blocked cells and one teleport pad on the ship. Blocked cells are shown in black, while the teleport pad is marked in red. The ship layout remains consistent across the "no bot," "optimal bot," and "learned bot" scenarios.



SETUP:

- The bot can move one step in any direction(up/down/left/right or across diagonals) or stay still.
- The crew member can only move in four cardinal directions.
- If the bot gets close to the crew member, it'll flee from the bot as far as possible in a random direction.
- If the bot isn't nearby, it'll just choose a random direction to move in.

BOT CHALLENGES:

Navigating the ship under normal circumstances is challenging enough, but with crew members in a state of panic, the task becomes more difficult. The erratic movements and unpredictable behavior of the crew member pose significant obstacles to the bot's objective.

SCENARIOS:

Within the scope of this project, we explore five distinct scenarios, each presenting a unique set of challenges and opportunities for the deployment of our security bot.

- **No Bot:** The crew member wanders aimlessly until it reaches the teleport pad in the absence of the bot.
- **Optimal Bot:** The Bot should take an optimal path to rescue the crew member to the teleport pad on the ship.
- **Learned Bot:** Build a machine learning model that takes the bot/crew configuration as the input and gives the optimal action as the output.
- **Generalizing:** The machine learning model is trained on various ship configurations.
- **Presence of Aliens:** The presence of one alien on the ship, the bot has to avoid the alien and try rescuing the crew member to reach the teleport pad for medical care.

DESIGN FRAMEWORK:

The project's design framework revolves around two core concepts: Value Iteration and Convolutional Neural Networks (CNNs).

1. Value Iteration

It serves as the primary method for optimizing decision-making processes, allowing the bot to refine its navigation strategy iteratively. Through this method, the bot continuously updates its understanding of the environment, assessing the value of different actions and adjusting its course accordingly. This iterative process ensures that the bot's decisions are optimized for guiding crew members to the teleport pad and help it get medical care.

The Bellman equation for value iteration

$$V(s) = \max_a (R(s, a) + \beta * \sum P(s' | s, a) V(s'))$$

Where,

- $V(s)$ is the value of state s .
- $R(s, a)$ is the immediate reward obtained from taking action 'a' in state 's'.
- $P(s' | s, a)$ is the probability of transitioning to state s' from state s after taking action a .
- β is the discount factor, representing the importance of future rewards.
- \max_a denotes the maximum over all possible actions 'a' from state 's'.

Pros of Value Iteration: It allows for continuous refinement of the bot's decision-making process, leading to improved navigation strategies over time. It is also easy to interpret and understand aiding in debugging.

Cons of Value Iteration: If the Bot's actions space increases, value iteration may become computationally expensive and time-consuming.

2. Convolutional Neural Network

A Convolutional Neural Network (CNN) is a deep learning algorithm designed primarily for image recognition and computer vision tasks. It consists of multiple layers, these layers include pooling layers that downsample feature maps, activation functions introducing non-linearity, and fully connected layers for classification or regression.

Pros of CNN: It is good in recognizing complex patterns and features within data, making them highly effective for interpreting the ship's layout and identifying optimal paths. CNNs also have the ability to learn from past experiences and adapt their strategies based on the observed pattern and yield an optimal result.

Cons of CNN: CNNs rely heavily on large volumes of training data to learn effectively, which may be challenging to obtain.

SCENARIO 1: NO BOT

We adopt a value iteration approach to compute T_{noBot} values for each cell on the ship. Initially, all T_{noBot} values on the cells of the ship are set to 0. The open cells adjacent to it are assigned a value of 1. Subsequently, through iterative updates, for every adjacent cell, we apply the previously derived value to calculate T_{noBot} . The value iteration process continues until convergence. The T_{noBot} (cell) value for the teleport pad cell is set to 0.

Algorithm:

Initialization:

- Create a matrix T_{noBot} of dimensions 11 X 11 initialized with max values (inf). This represents the expected steps to reach the teleport pad from each cell.
- Set T_{noBot} as 0 for the teleport pad as it is the target endpoint.

Value Iteration process:

- This process uses a value iteration technique to compute the number of steps for a crew to reach the teleport pad.
- For every crew position, explore all possible movements within the ship.
- Calculate the expected number of steps for a crew to reach the teleport pad from each open cell using the bellman equation.
- Repeat the process until the expected steps stabilize. Continue updating T_{noBot} until there are no significant changes between iterations.

For each (*crew*) position:

For c' in crew actions:

$$\text{sum} += \frac{1}{\text{len}(\text{neighbors})} * \sum T_{noBot}(c')$$

$$T_{noBot}(c'_{\text{new_value}}) = 1 + \text{sum}$$

What is T_{noBot} (cell) in relation to the value of T_{noBot} for its neighbors? Give an explicit equation.

The expected number of steps required by the crew C_x to reach the teleport pad is given by bellman equation:

$$T_{noBot}(C_x) = 1 + \frac{1}{\text{Neighbors of the cell } C_x} \times \sum_{C_{ny}} T_{noBot}(C_{ny})$$

where,

- C_{ny} represents the valid neighbors of the crew.

Solve for the value of T_{noBot} for each cell. (Hint: T_{noBot} (teleport pad) = 0)

Initially assign all the T_{noBot} (open cell) to zero and now update the T_{noBot} value of the adjacent cells of the teleport pad to 1. Then, for each neighboring cell, by using the above equation, compute T_{noBot} iteratively until convergence.

T_no_bot											
	0	1	2	3	4	5	6	7	8	9	10
0	226.47250	225.37539	222.43672	218.06075	211.69383	208.70165	0.0	217.52734	222.66711	225.81635	226.87240
1	225.58928	224.23634	220.89293	0.0	205.33668	203.71806	205.17525	210.40575	0.0	224.92840	225.93784
2	223.07843	221.10683	213.01485	203.07221	197.61508	191.67533	198.41904	205.52323	211.70414	220.04952	223.02192
3	219.55820	0.0	203.00467	195.60328	186.39193	162.96422	187.31867	197.58093	206.55733	216.56175	220.08743
4	214.04714	206.26220	197.41247	185.95994	0.0	82.482113	0.0	186.94072	196.39945	0.0	217.68749
5	213.33890	204.34399	190.43887	161.87080	81.935404	0.0	82.924909	163.85650	192.70815	206.04927	213.29620
6	218.64353	0.0	194.14311	185.16007	0.0	83.257690	0.0	188.86779	200.54391	209.16057	213.16044
7	221.95759	217.93034	203.84659	196.47446	187.90217	164.52210	189.16280	199.21918	207.45599	212.90584	214.03314
8	226.31767	225.00483	0.0	205.00524	199.72489	193.78138	200.76337	207.40694	213.17222	216.99097	0.0
9	229.00974	227.77604	224.19922	215.83349	208.22723	206.13169	208.71920	212.49017	216.85241	221.90373	224.89475
10	229.95472	228.90927	226.00643	221.91996	0.0	209.81591	211.50866	212.99942	0.0	224.89475	225.89475

The table presented above shows the average number of steps it takes for each cell to reach the teleport pad. For example, if the crew is at (5,2), the average steps needed to reach the teleport pad is approximately 190.4388.

Bonus: Why don't we need a β ?

The discount factor β isn't required because the crew member moves randomly across the ship in the absence of the Bot. We can solely focus on immediate rewards without considering any future consequences. Random movement of the crew doesn't impact future rewards and no long-term consequences. Therefore the movement of the crew member isn't influenced by any factor and hence, the discount factor isn't required.

Are there any cells with an infinite expected time? Why or why not?

Yes, if the crew is isolated by blocked cells, it won't be able to reach the teleport pad. In such cases, the expected time for the crew to reach the teleport pad will be infinite. Additionally, blocked cells have an infinite expected time.

If the crew member is initially randomly chosen cell (uniform), what is the expected time to get to the teleport pad?

To determine the expected time to reach the teleport pad from a randomly chosen cell, you can calculate the average expected time by summing the expected times from all cells and dividing by the total number of open cells.

SCENARIO 2: OPTIMAL BOT

We adopt a value iteration approach to compute T_{Bot} values for each cell on the ship. Initially, all T_{Bot} values on the cells of the ship are set to high default values to represent uninitialised states. The T_{Bot} (bot cell, crew cell) value for the teleport pad cell is set to 0. We then iteratively update the values, considering all the possible moves the bot can take and averaging the expected cost based on all possible crew moves. The process continues until the values converge, indicating the optimal expected cost for each bot-crew position on the ship.

Algorithm:

1) Expected time for crew to reach the teleport pad ($T_{bot_timestamp}$):

Initialization:

- Define a dictionary variable T_{Bot} to store the expected time for the crew to reach the teleport pad from each open cell in the ship.
- Initialize T_{Bot} with high values to represent uninitialised states.
- Set T_{Bot} as 0 for the teleport pad as it is the target endpoint.

Value Iteration process:

- The process involves value iteration technique to update the expected times for each bot-crew position pair.
- For each possible bot and crew position, explore all the possible valid actions of the bot. Every bot has 9 actions and for each bot position, there are 4 crew actions possible.
- Determine the minimum expected time for the bot to guide the crew to the teleport pad, considering all valid actions of the bot using the bellman's equation.

For each $(bot, crew)$ position:

For b' in bot actions:

For c' in crew actions:

$$sum += \frac{1}{len(neighbors)} * T_{bot}(b', c')$$

$$b'_{value} = 1 + sum$$

$$T_{bot}(bot, crew) = \min (b'_{value} \text{ computed for all 9 bot actions})$$

- Convergence is achieved when no significant changes occur in the expected time.

2) Determine Optimal bot actions to guide crew:

Initialization:

- Define dictionary variables T_{Bot} and T_{Bot_Action} to store the expected time for the crew to reach the teleport pad and the optimal actions for the bot respectively.
- Initialize T_{Bot} with high values to represent uninitialised states and set T_{Bot} as 0 for the teleport pad.
- Initialize T_{Bot_Action} with default bot positions.

Value Iteration process:

- The process involves a similar value iteration process through all the possible bot-crew positions to find the optimal bot movements.
- For each possible bot and crew position, explore all the possible valid actions of the bot.
- For every valid action the bot takes, we'll calculate the reward for the Bellman equation by negating the values from T_{Bot} to find the maximum possible reward.
- Calculate the expected time for the current state as the sum of the “reward” and the average expected time of the crew moves.
- If this new expected time is lower than the current value, update the expected time and record the optimal bot move.

For each $(bot, crew)$ position:

For b' in bot actions:

Calculate *reward*

For c' in crew actions:

$$sum += \frac{1}{len(neighbors)} * T_{bot}(b', c')$$

$$b'_{value} = reward + sum$$

$$T_{bot_action}(bot, crew) = argmax(b'_{value} \text{ computed for all 9 bot actions})$$

- Convergence is achieved when no significant changes occur in the expected time.

What is $T_{Bot}(bot \text{ cell}, crew \text{ cell})$ in relation to the available actions / accessible cells? Give an explicit equation.

The expected time (**$T_{bot_timestamp}$**) for the bot B_x to guide the crew C_x to reach the teleport pad at is given by the equation:

$$T_{Bot}(B_x, C_x) = \min_{B_{ny}} (1 + \sum_{C_{ny}} P(C_{ny} | C_x, B_{ny}) * T_{Bot}(B_{ny}, C_{ny}))$$

Where,

- B_{ny} represents the valid actions of the bot.
- C_{ny} represents the valid neighbors of the crew.
- $P(C_{ny} | C_x, B_{ny})$ is the transition probability from the state C_x to C_{ny} which is calculated as:

$$P(C_{ny} | C_x, B_{ny}) = \frac{1}{\text{Number of open neighbors for crew } (C_{ny})}$$

Optimal Policy Equation:

The optimal policy describes the best action for the bot to guide the crew toward the teleport pad, derived from the expected times calculated in T_{Bot} as:

$$T_{Bot}(B_x, C_x) = \operatorname{argmax} (\text{Reward} + \sum_{C_{ny}} P(C_{ny} | C_x, B_{ny}) * T_{Bot}(B_{ny}, C_{ny}))$$

Where,

- B_{ny} represents the valid actions of the bot.
- C_{ny} represents the valid neighbors of the crew.
- The ‘Reward’ is determined from the ‘ T_{Bot} ’, which represents the immediate cost of moving from the bot’s current position to the next valid action. To maximize the value according to the Bellman equation, we use the negation of ‘ T_{Bot} ’.
- $P(C_{ny} | C_x, B_{ny})$ is the transition probability from the state C_x to C_{ny} which is calculated as:

$$P(C_{ny} | C_x, B_{ny}) = \frac{1}{\text{Number of open neighbors for crew } (C_{ny})}$$

Solve for the value of T_{Bot} for each cell pair, and note the optimal action to take in each case.

Taking the bot position as (0,0) and for every valid crew position in the ship, the T_{Bot} can be given as:

T_Bot for Bot_pos = (0,0) and every other Crew Position

	0	1	2	3	4	5	6	7	8	9	10
0	inf	32.118433	29.770805	30.235962	28.699164	29.224854	inf	34.366168	36.227515	37.511792	37.951589
1	32.209606	31.147597	29.426347	inf	28.254850	28.266264	29.715769	31.808735	inf	37.340160	37.709212
2	29.993947	29.537918	28.269124	26.604069	25.938400	26.232398	28.343727	30.822042	33.054626	35.856260	36.871612
3	30.600309	inf	26.553588	24.711635	24.440908	22.083203	26.468417	29.459147	32.155524	34.888441	36.154239
4	29.206670	28.301103	25.705657	24.264057	inf	11.296204	inf	28.639343	30.277057	inf	35.697773
5	30.656108	27.929956	25.721547	21.717015	11.106649	0	13.431389	25.574212	30.443912	33.077302	34.787124
6	32.465519	inf	27.119766	25.700441	inf	13.401609	inf	29.901434	31.902691	33.846783	34.728529
7	34.547399	33.556040	29.950669	28.864764	28.511900	25.535728	29.800808	31.624241	33.477877	34.785126	34.995211
8	35.957278	35.860645	inf	31.325421	30.621343	30.263446	31.695085	33.313419	34.898273	36.014557	inf
9	37.145422	36.697079	36.085044	33.924487	32.553432	32.426270	33.252847	34.427251	35.876052	37.744055	38.712639
10	37.307523	37.276554	36.400668	35.543489	inf	33.298428	33.848815	34.290765	inf	38.712639	39.198088

Here for the bot position and all blocked cells the value of T_{Bot} is inf and for teleport pad the value of T_{Bot} is zero.

The optimal action for the bot located at coordinates (0,0) and for each crew position can be outlined as below :

Actions for Bot_pos = (0,0) and every other Crew Position

	0	1	2	3	4	5	6	7	8	9	10
0	(-1, -1)	(1, 0)	(0, 1)	(1, 1)	(1, 1)	(0, 1)	(-1, -1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)
1	(0, 1)	(1, 0)	(1, 1)	(-1, -1)	(0, 1)	(1, 1)	(1, 1)	(1, 1)	(-1, -1)	(1, 1)	(1, 1)
2	(1, 0)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)
3	(1, 1)	(-1, -1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)
4	(1, 1)	(1, 0)	(1, 1)	(1, 1)	(-1, -1)	(1, 1)	(-1, -1)	(1, 1)	(1, 1)	(-1, -1)	(1, 1)
5	(1, 0)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(0, 0)	(1, 1)	(1, 1)	(1, 1)	(0, 1)	(1, 1)
6	(1, 0)	(-1, -1)	(1, 0)	(1, 1)	(-1, -1)	(1, 1)	(-1, -1)	(1, 1)	(1, 1)	(0, 1)	(0, 1)
7	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(0, 1)
8	(1, 0)	(1, 0)	(-1, -1)	(1, 0)	(1, 0)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(-1, -1)
9	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(1, 1)	(1, 1)	(1, 1)	(1, 1)	(1, 1)
10	(1, 0)	(1, 0)	(1, 0)	(1, 0)	(-1, -1)	(1, 0)	(1, 0)	(1, 1)	(-1, -1)	(1, 1)	(1, 1)

In this context, each cell's coordinates represent the optimal action the bot should take for each crew position on the ship. The actions in blocked cells are denoted as (-1,-1).

Are there any pairs where the optimal action for the bot is to just stay still?

In cases where moving the bot would eventually block the crew from any further movement, the optimal move for the bot would be to stay still. If the time taken by the crew to reach the teleport pad, when the bot moves to possible neighboring cells, is more than the time taken when the bot stays in the same position, then the bot stays in its position.

How do the ‘time to escape’ values for the no-bot situation compare with time to escape with the bot? How much improvement is there, and what configurations give very little improvement?

T_no_bot											
	0	1	2	3	4	5	6	7	8	9	10
0	226.47250	225.37539	222.43672	218.06075	211.69383	208.70165	0.0	217.52734	222.66711	225.81635	226.87240
1	225.58928	224.23634	220.89293	0.0	205.33668	203.71806	205.17525	210.40575	0.0	224.92840	225.93784
2	223.07843	221.10683	213.01485	203.07221	197.61508	191.67533	198.41904	205.52323	211.70414	220.04952	223.02192
3	219.55820	0.0	203.00467	195.60328	186.39193	162.96422	187.31867	197.58093	206.55733	216.56175	220.08743
4	214.04714	206.26220	197.41247	185.95994	0.0	82.482113	0.0	186.94072	196.39945	0.0	217.68749
5	213.33890	204.34399	190.43887	161.87080	81.935404	0.0	82.924909	163.85650	192.70815	206.04927	213.29620
6	218.64353	0.0	194.14311	185.16007	0.0	83.257690	0.0	188.86779	200.54391	209.16057	213.16044
7	221.95759	217.93034	203.84659	196.47446	187.90217	164.52210	189.16280	199.21918	207.45599	212.90584	214.03314
8	226.31767	225.00483	0.0	205.00524	199.72489	193.78138	200.76337	207.40694	213.17222	216.99097	0.0
9	229.00974	227.77604	224.19922	215.83349	208.22723	206.13169	208.71920	212.49017	216.85241	221.90373	224.89475
10	229.95472	228.90927	226.00643	221.91996	0.0	209.81591	211.50866	212.99942	0.0	224.89475	225.89475

T_bot values for crew_pos = (8,3)											
	0	1	2	3	4	5	6	7	8	9	10
0	inf	32.118433	29.770805	30.235962	28.699164	29.224854	inf	34.366168	36.227515	37.511792	37.951589
1	32.209606	31.147597	29.426347	inf	28.254850	28.266264	29.715769	31.808735	inf	37.340160	37.709212
2	29.993947	29.537918	28.269124	26.604069	25.938400	26.232398	28.343727	30.822042	33.054626	35.856260	36.871612
3	30.600309	inf	26.553588	24.711635	24.440908	22.083203	26.468417	29.459147	32.155524	34.888441	36.154239
4	29.206670	28.301103	25.705657	24.264057	inf	11.296204	inf	28.639343	30.277057	inf	35.697773
5	30.656108	27.929956	25.721547	21.717015	11.106649	0	13.431389	25.574212	30.443912	33.077302	34.787124
6	32.465519	inf	27.119766	25.700441	inf	13.401609	inf	29.901434	31.902691	33.846783	34.728529
7	34.547399	33.556040	29.950669	28.864764	28.511900	25.535728	29.800808	31.624241	33.477877	34.785126	34.995211
8	35.957278	35.860645	inf	31.325421	30.621343	30.263446	31.695085	33.313419	34.898273	36.014557	inf
9	37.145422	36.697079	36.085044	33.924487	32.553432	32.426270	33.252847	34.427251	35.876052	37.744055	38.712639
10	37.307523	37.276554	36.400668	35.543489	inf	33.298428	33.848815	34.290765	inf	38.712639	39.198088

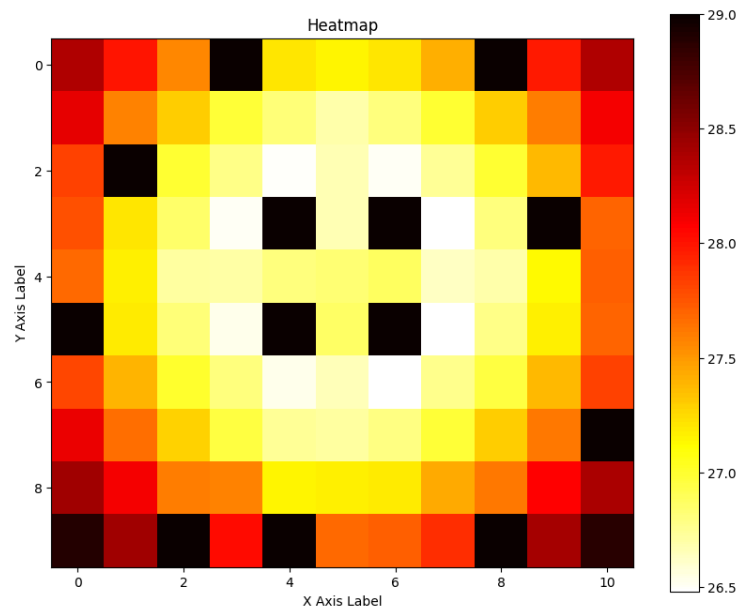
In the T_no_bot scenario, the crew takes an average of 205.0052 steps to reach the teleport pad positioned at (8,3). Conversely, in the T_bot scenario, regardless of where the bot is placed on

the ship, the crew's average steps to reach the teleport pad are significantly lower. For instance, if the bot is positioned at (3,0), the crew averages 30.60 steps, indicating a substantial difference of about 174 steps.

In every configuration, the difference between the `T_no_bot` and `T_bot` scenarios in terms of the average steps taken by the crew is very large. There is no arrangement, except for the teleport pad, that leads to only a little improvement, as shown in the tables above.

If the bot gets to pick where it starts, and then the crew member will be placed at random in the remaining cells (uniformly), where should the bot choose to start, and why?

If the bot has a chance to pick its starting position, it chooses the cells adjacent to the fixed blocked cells near the teleport pad, as indicated by the heatmap below. These cells are derived by calculating, for each bot position, the average of time steps taken by the crew from all other cells on the ship. The minimum values from the above result are shown in white in the heatmap. The bot can be placed in the minimum value cell in order to guide the crew to the teleport pad in lesser time steps.



How does the time to escape compare in this case to the randomly placed crew situation previously?

When the bot is placed in an optimal position initially, the time taken is calculated as the average of time taken by crew in all the other positions with respect to that bot position in the ship. In the previous case, the crew is placed randomly in any position without any bot on the ship. Due to the influence of the bot, these time steps will be less when compared to the time taken by the crew in no bot case. Therefore the average time taken when placing bot in optimal position is lesser compared to the case where crew is placed randomly without bot.

SCENARIO 3: LEARNED BOT

Having identified the optimal actions for various bot/crew configurations in the previous scenario, we now train a machine learning model to automate these decisions. In this scenario, our objective is to develop a model that takes a bot/crew configuration as input and outputs the optimal action for the bot to take in that specific scenario.

How are you structuring your model? How are you structuring your data?

A sequential feedforward neural network is used for structuring the model. The neural network used in this project is a convolutional neural network. The Convolutional Neural Network is utilized specifically for training a model to determine the optimal actions for a bot based on different configurations of bot and crew. Prior to this scenario, for a fixed configuration of bot and crew, the average step taken by the bot to help the crew member reach the teleport pad was determined. Now, the focus shifts to training a model that takes as input a fixed bot/crew configuration and outputs the optimal action for the bot to take in that specific configuration. This necessitates the utilization of a CNN architecture tailored to process and learn from the input configurations effectively.

The neural network architecture comprises three Dense layers, which consists of fully connected layers where each neuron is connected to every neuron in the preceding and succeeding layers. These Dense layers facilitate the extraction of abstract features from the input data, enabling the network to learn complex relationships in obtaining the optimal path for the bot to rescue the crew member.

The Input layer consists of 4 units that correspond to the input shape of the data. Each unit represents the Bot_x, Bot_y, Crew_x and Crew_y attributes of the input data, and collectively

they form the initial representation of the input provided to the neural network. This layer serves as the entry point for the data in the network and sets the dimensionality for subsequent layers.

The Hidden layer is sandwiched between the Input and Output layer. We have built a single hidden layer consisting of 128 units. The Rectified Linear Unit (ReLU) activation function is applied to the output of each neuron in this layer to introduce non-linearity. It operates by outputting the input value if it's positive, and zero otherwise, represented mathematically as $f(x)=\max(0,x)$. ReLU effectively mitigates the vanishing gradient problem leading to faster convergence during training.

The Output layer, the final component of the neural network, has 2 units. These units generate the final output of the network, which corresponds to the Action_x and Action_y that predicted optimal actions for the bot. The activation function used in this layer is a Linear activation function, enabling the network to output continuous values for the optimal actions the bot has to take without constraining them to a specific range.

For the input, the information about the current positions of both the bot and the crew. This includes the coordinates (x, y) of the bot and crew members within the ship. These coordinates provide spatial information necessary for the model to make decisions about the optimal action for the bot.

The output component of the data represents the action that the bot should take in response to the given input configuration. This action is also represented by coordinates (action_x, action_y), indicating the direction in which the bot should move next to guide the crew member towards the teleport pad.

To structure the data, we have organized it into pairs, where each pair contains an input configuration and its corresponding output action. A single data point in the dataset might look like this: ((bot_x, bot_y, crew_x, crew_y), (action_x, action_y)). These pairs form the dataset used for training the machine learning model. Each pair captures a specific scenario or configuration of the bot and crew within the grid.

By structuring the data in this way, the necessary information for the model to learn the relationship between different bot/crew configurations and the corresponding optimal actions is enabled. Thereby, the model generalizes its learning and makes effective decisions in guiding the crew towards the teleport pad across various scenarios.

How are you training your model?

Training the model involves feeding the structured dataset into a machine learning algorithm and optimizing its parameters to learn the mapping between input configurations and optimal actions.

The first step in training is to prepare the data. This involves organizing the dataset into training and test sets. The training set serves as the foundation for teaching the model, while the test set aids in assessing the model's performance and prevents overfitting.

The CNN model architecture is designed to handle the spatial nature of the input data. In this scenario, the iterative process involves an epoch value of 200 with feeding batches of size 32 input-output pairs into the model and adjusting its parameters to minimize the error between the predicted actions and the true optimal actions. Optimization algorithm used is Adam that minimizes the loss, gradually improving its performance.

Validation plays an important role throughout the training process. After each epoch, the model's performance is evaluated on the test set. This allows for monitoring the model's generalization ability and detecting signs of overfitting.

Training continues until the stopping criterion is met, such as reaching a maximum number of epochs or observing no improvement in validation performance over several iterations. Through this iterative process, the model learns to accurately predict the optimal actions for guiding the crew to the teleport pad in various configurations of the fixed ship layout.

How can you tell how accurate your model is?

In assessing the performance of the CNN model, the Mean Absolute Error (MAE) metric was employed as the primary evaluation criterion. MAE is a measure of the average absolute difference between the predicted actions of the bot and the true optimal actions. Mathematically, it is defined as:

$$MAE = \frac{1}{n} \sum_{i=1}^n | y_i - y_i' |$$

Where,

n - the number of samples in the testing set.

y_i - true optimal action for sample i

y_i' - predicted action by the model for sample i

During the evaluation process, the CNN model was trained iteratively over epochs equal to 100, with the MAE computed after each epoch on the testing set. The MAE value served as a quantitative measure of the model's accuracy in predicting the optimal actions for guiding the crew member to the teleport pad.

How can you avoid overfitting? Do you need to? Why or why not?

To ensure the CNN model effectively learns to predict the optimal actions for the bot in various configurations of bot and crew, it's important to address the issue of overfitting. Overfitting occurs when the model learns to memorize the training data rather than generalize from it, leading to poor performance on unseen data. In our project, overfitting doesn't occur because of the size of the input data being fed to the neural network. A larger data set with a lot of constraints would lead to overfitting the neural network

Does accuracy of your model translate to effectiveness of a bot based on your model (i.e., can it herd the crew member to the teleport)? Experiment, compare, and verify. Explain your results.

The accuracy of the model doesn't translate to the effectiveness of a bot based on our model. This is so because the accuracy measure is based on the prediction of the model whereas the effective measure measurement talks about the success rate that is based on the average steps taken by the bot in rescuing the crew or the the time taken for multiple runs. The average value of steps taken by the bot is in the range of 35 to 40. The accuracy of the neural network is 96.13%. The comparison can be made by comparing the values obtained in the optimal bot scenario.

If your initial model fails to translate to an effective bot - try again, and build a model that can be used to guide the bot in place of the computed policy from the previous section. Is your model smaller or larger in memory than the lookup table you computed in the previous section?

The memory size of the lookup table can be calculated as:

$$\text{Memory size} = \text{Number of integers stored}$$

We need to convert the integers to kilobytes (KB).


$$\text{Memory size} = 4 * \text{Number of integers stored}$$

Here,

$$\text{Number of integers stored} = ((11 \times 11 - \text{No of blocked cells}) * (11 \times 11 - \text{No of blocked cells}))$$

The number of blocked cells in our ship is 14. So, the number of integers stored is $107 * 107$ bytes. So, the memory size is

$$\text{Memory size} = 4 * 107 * 107 = 45796 = 45.796 \text{ KB}$$

Name	Size	Kind
 trained_learned_bot.h5	134 KB	Document

Here, the memory size of the learned bot is lesser than the memory size of the lookup table with a difference of 88 KB.

Initial Idea:

Initially, we opted to implement Support Vector Machines (SVMs) for guiding the bot to rescue crew members and lead it safely to the teleport pad amidst the chaos of an alien invasion. However, as we proceeded deeper into the experimentation phase, several challenges emerged, highlighting the issues of SVMs.

One of the primary hurdles we encountered was the complexity of the ship's layout, characterized by nonlinear relationships and spatial data. SVMs, by design, assume linear separability, which means they excel in scenarios where data can be neatly divided by a linear boundary. As a result, SVMs struggled to capture the spatial relationships necessary for effective guidance for the bot to rescue the crew member.

Moreover, SVMs exhibited a pronounced sensitivity to noise and outliers in the data. SVMs, lacking robust mechanisms to mitigate the impact of such disturbances, struggled to discern meaningful patterns amidst the noise, resulting in unreliable performance.

Ultimately, the combination of SVMs' computational inefficiency, susceptibility to overfitting, and vulnerability to noise rendered the performance of guiding the bot to rescue crew members. Recognizing the urgent need for a more adaptive and robust approach, we went ahead with convolutional neural networks.

SCENARIO 4: GENERAL BOT

Try to build and train a model capable of guiding the bot regardless of what ship configuration it is on. How are you approaching this problem? Where are you getting your data from? How can you assess the quality and effectiveness of your model?

In this scenario, our objective is to train a model capable of guiding the bot to rescue crew members across 20 distinct ship layouts. Each layout presents unique configurations of bots and crew members, posing challenges that require the bot to adapt its actions accordingly. Therefore, we employ a convolutional neural network (CNN) architecture, chosen for its ability to handle spatial data effectively.

The dataset used for training is sourced from a CSV file containing organized pairs of input configurations and corresponding output actions. These configurations encompass the diverse layouts of ships, with variations in the positions of bots, crew members, and obstacles. Each pair provides crucial information about the spatial relationships within the layout and the optimal actions required for successful rescue missions.

Training the CNN model involves feeding batches of input-output pairs to the network and optimizing its parameters to minimize the discrepancy between predicted and true optimal actions. The training process spans 200 epochs, allowing the model to learn and adapt to the intricacies of each ship layout. A batch size of 32 is selected to balance computational efficiency and model convergence. The Adam optimizer is utilized to facilitate efficient gradient descent, thereby enhancing the model's ability to generalize across different layouts.

The neural network architecture comprises an input layer, a hidden layer, and an output layer. The input layer consists of four units, representing the spatial attributes of the bot and crew members within each ship layout. These attributes serve as the foundation for the subsequent layers, providing essential information for decision-making. The hidden layer, containing 128 units, utilizes the ReLU activation function to introduce non-linearity and address the vanishing gradient problem, ensuring effective learning from spatial data. The output layer consists of two units, generating the predicted optimal actions for the bot. By employing a Linear activation function, the model can produce continuous output values, enabling precise guidance across different ship layouts.

To evaluate the model's accuracy and effectiveness, we rely on the Mean Absolute Error (MAE) metric. MAE quantifies the average absolute difference between predicted and true optimal actions, achieving a 97% performance across diverse ship layouts.

CONTRIBUTIONS

Puja Sridar

- Worked on implementing no bot scenario.
- Testing and Validation.
- Worked on the project report.

Vaishnavi Madhavaram

- Worked on implementing an optimal bot scenario.
- Testing and Validation.
- Worked on the project report.

Kondaveti Deviram

- Worked on implementing learned bot and generalization scenarios.
- Testing and Validation.
- Worked on the project report.