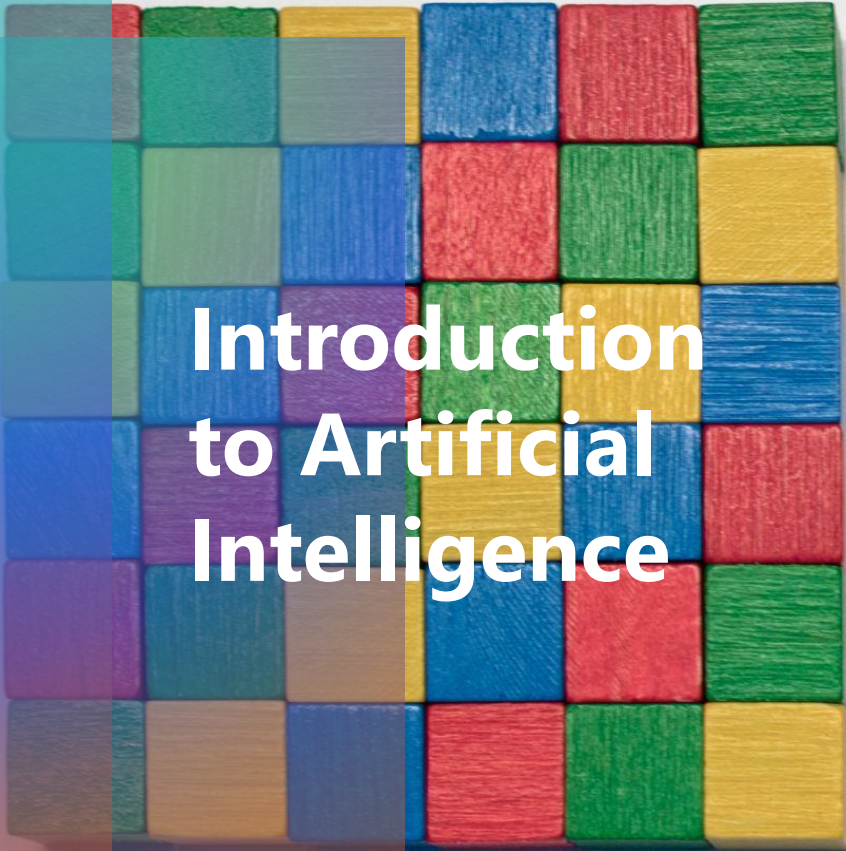


Feature-Based Reinforcement Learning for the Rubik Cube

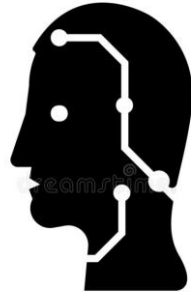
Team Members:

1. Alisha Kareemulla
2. Thulasi Theja K S

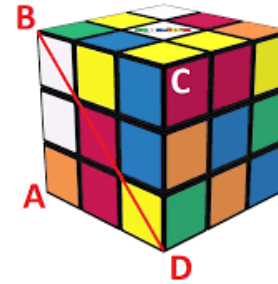
A 3x3 Rubik's Cube is shown, partially solved, with various colored squares visible. The text "Introduction to Artificial Intelligence" is overlaid on the center of the cube.

Introduction
to Artificial
Intelligence

Description of Project



Create and train a reinforcement learning agent that learns to solve a 2x2x2 and 3x3x3 size Rubik cube



This is done using feature-based state representations

Project Objectives



- ❖ Generating the initial cube and scrambling it randomly every time.
- ❖ Creating a RL agent that can solve 2x2x2 and 3x3x3 rubric cube.
- ❖ Training the agent based on the state of the cube using Reinforcement learning algorithms and comparing them.

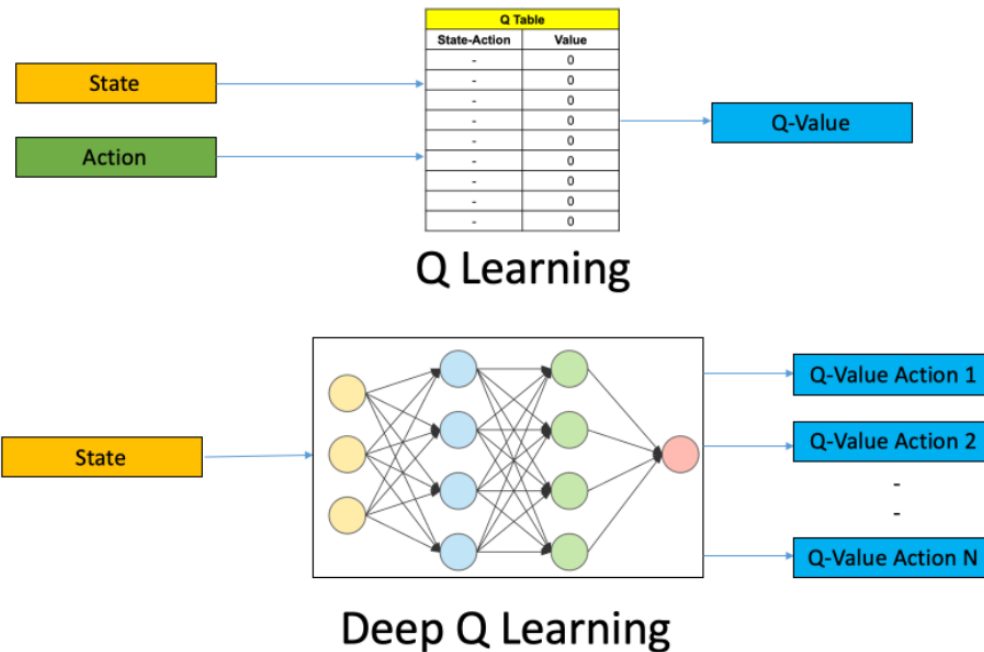
Approach



- ❖ Creating the rubric cube design for 2x2x2 and 3x3x3 using Python and its modules.
- ❖ Creating an agent that can solve the cube.
- ❖ Training the agent based on the state of the cube.
- ❖ Reinforcement learning algorithms i.e., Deep Q-Learning and SARSA is used.
- ❖ The reward system primarily focuses on the number of entirely-correct blocks / current state based on the algorithm.

Deep Q-Learning

- ❖ Involves Agent-> a
set of states-> S
set of Actions per state-> A
reward, r
epsilon
Discount factor, gamma -> probability to succeed at every step -> between 0 and 1



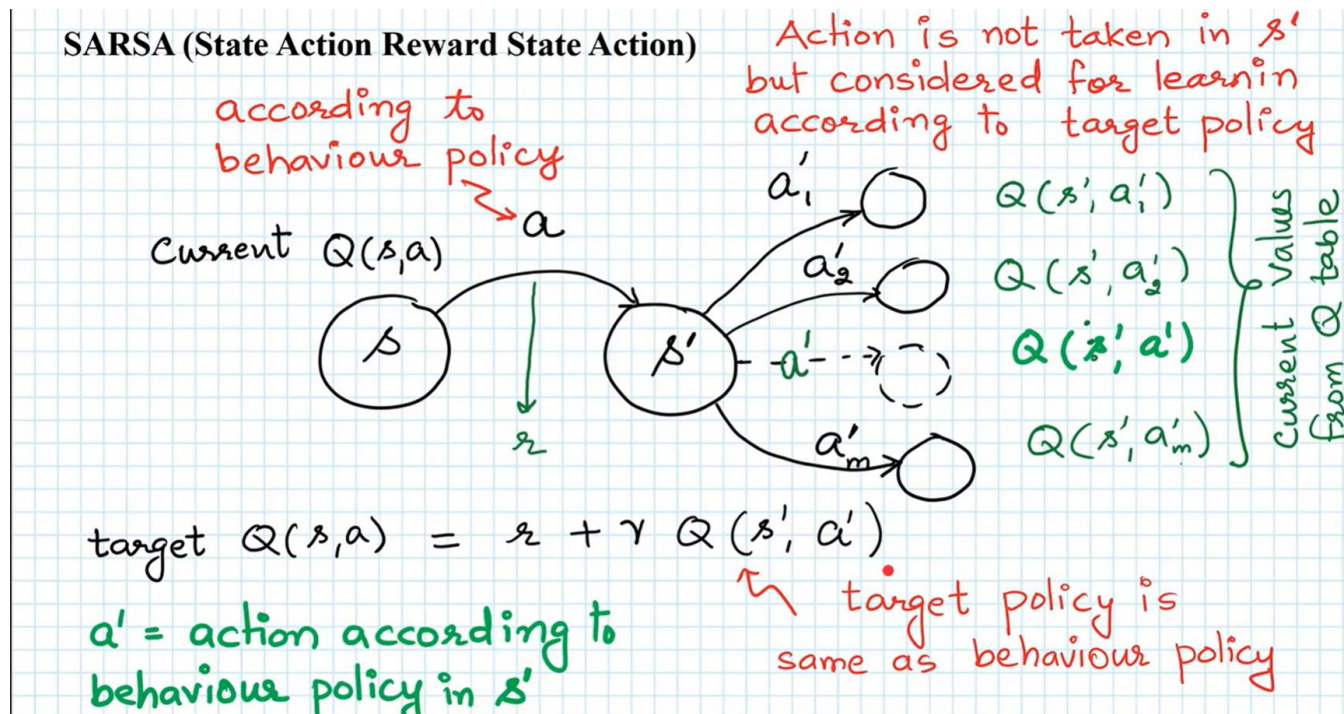
Procedure

- ❖ All the past experience is stored by the user in memory.
- ❖ The next action is determined by the maximum output of the Q-network.
- ❖ The loss function here is mean squared error of the predicted Q-value and the target Q value – Q^* .
- ❖ However, we do not know the target or actual value here as we are dealing with a reinforcement learning problem.
- ❖ The Q-value update equation derived from the Bellman equation

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]$$

SARSA

- ❖ State-action-reward-state-action (SARSA) is an algorithm for learning a Markov Decision process policy.
- ❖ Agent interacts with the environment and updates the policy based on actions taken, hence this is known as an on-policy learning algorithm



Final Hyperparameter Values:

- Discount (Gamma): 0.9
- Learning Rate: 0.05
- No. of Episodes: 100
- Epsilon: 0.2

Q – Learning vs SARSA (State Action Reward State Action) Algorithm

Q – Learning (Off policy)

Updated Q Value Current Q Value Target Q Value Current Q Value

$$\overset{\text{Updated Q Value}}{\dot{Q}(s, a)} = \overset{\text{Current Q Value}}{Q(s, a)} + \alpha \left[r + \underbrace{\max_{a'} \gamma Q(s', a')}_{\text{Target Q Value}} - \overset{\text{Current Q Value}}{Q(s, a)} \right]$$

α = Learning Rate

Target policy is always Greedy Policy

SARSA (State Action Reward State Action) Algorithm (On policy)

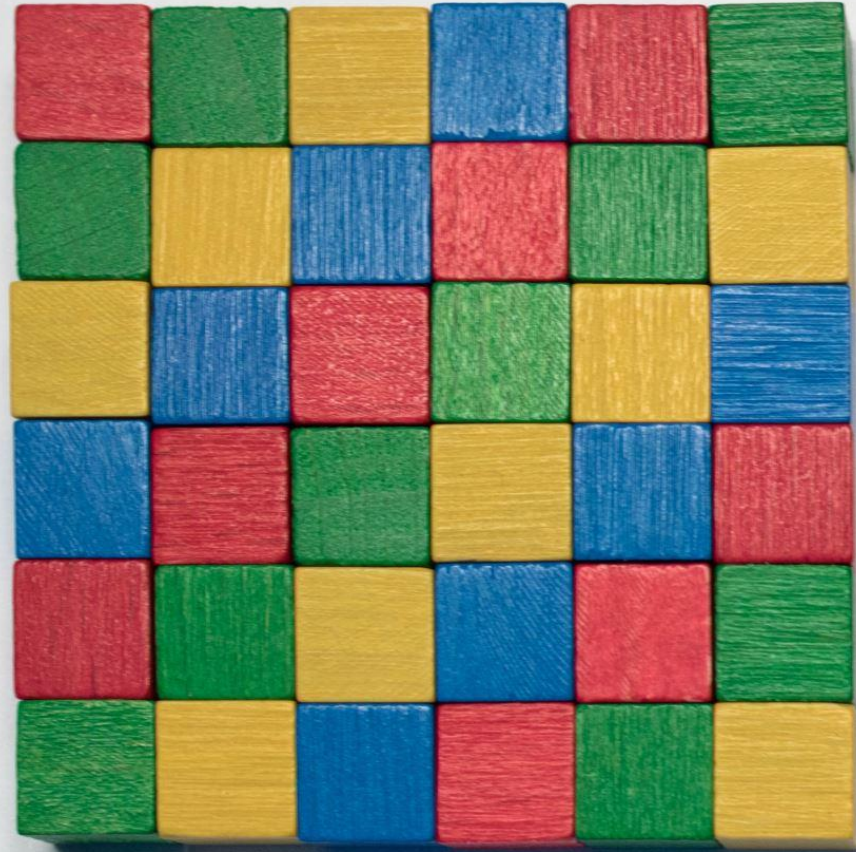
Updated Q Value Current Q Value Target Q Value Current Q Value

$$\overset{\text{Updated Q Value}}{Q(s, a)} = \overset{\text{Current Q Value}}{Q(s, a)} + \alpha \left[r + \underbrace{\gamma Q(s', a')}_{\text{Target Q Value}} - \overset{\text{Current Q Value}}{Q(s, a)} \right]$$

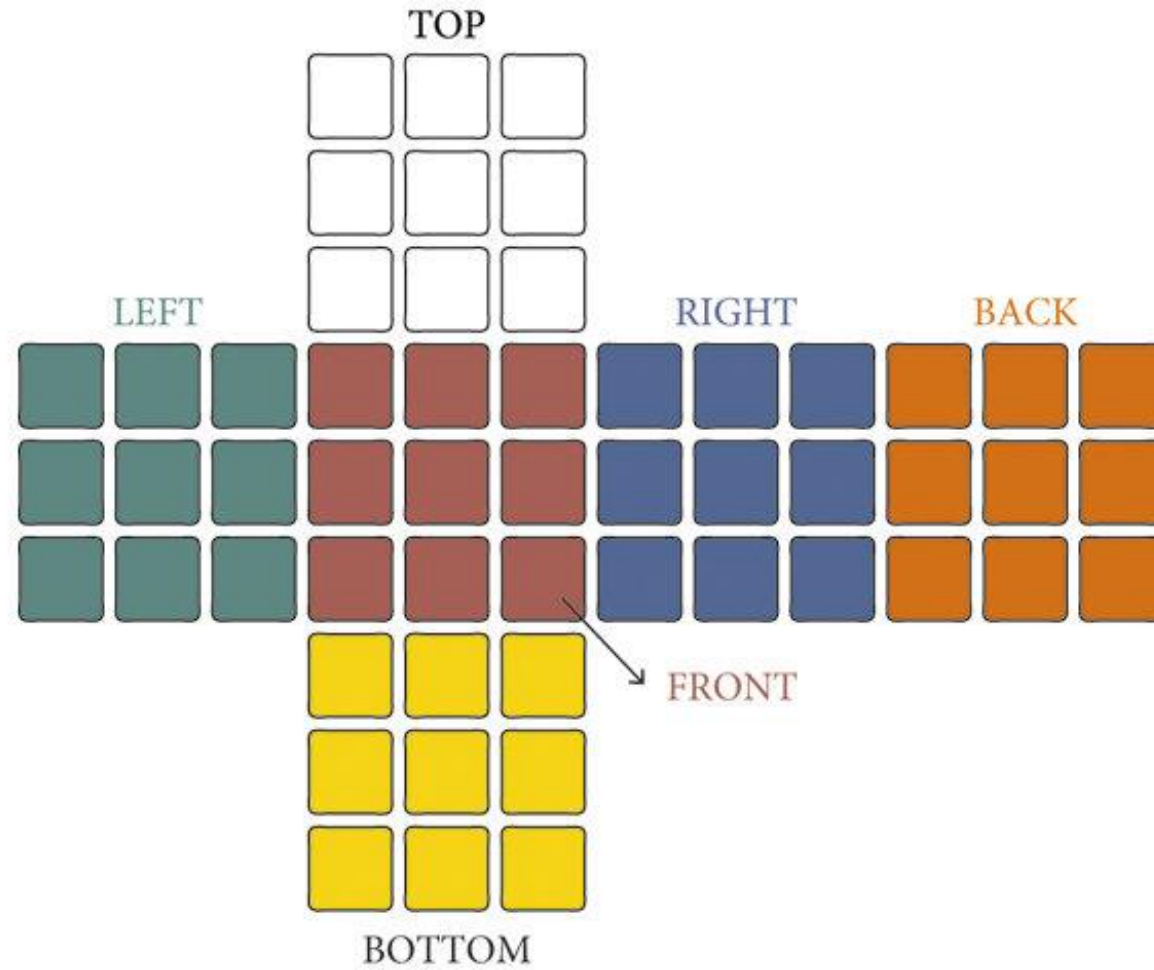
α = Learning Rate

Target Policy is always same as
Behaviour Policy

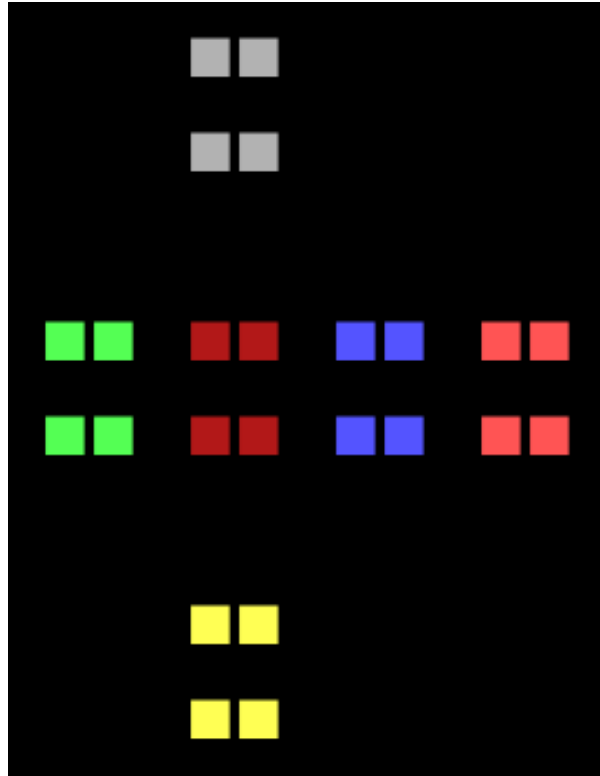
Code Review



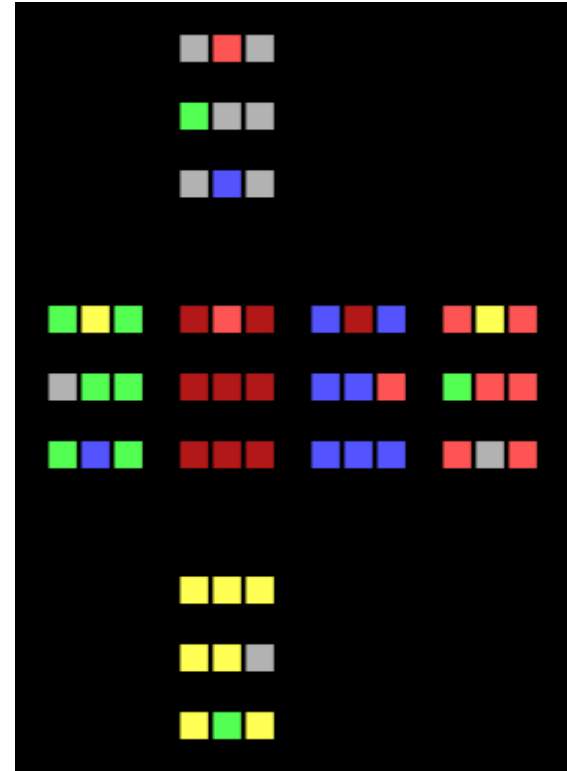
Cube Representation



Deep Q-Learning Stats



24 solved cubes in 45 min of training
2x2x2



38 solved cubes in 3.5 hours of training
3x3x3

2x2x2 cube stats after 45 minutes of training

Cube 1034	(Attempt 1037):	Rotations:	8781,	Time:	3
Cube 1035	(Attempt 1038):	Rotations:	5937,	Time:	2
Cube 1036	(Attempt 1039):	Rotations:	1591,	Time:	0
Cube 1037	(Attempt 1040):	Rotations:	11969,	Time:	5
Cube 1038	(Attempt 1041):	Rotations:	1615,	Time:	0
Cube 1039	(Attempt 1042):	Rotations:	77,	Time:	0
Cube 1040	(Attempt 1043):	Rotations:	6189,	Time:	2
Cube 1041	(Attempt 1044):	Rotations:	13945,	Time:	5
Cube 1042	(Attempt 1045):	Rotations:	497,	Time:	0
Cube 1043	(Attempt 1046):	Rotations:	9313,	Time:	3
Cube 1044	(Attempt 1047):	Rotations:	13473,	Time:	5
Cube 1045	(Attempt 1048):	Rotations:	353,	Time:	0
Cube 1046	(Attempt 1049):	Rotations:	649,	Time:	0
Cube 1047	(Attempt 1050):	Rotations:	4187,	Time:	1
Cube 1048	(Attempt 1051):	Rotations:	4793,	Time:	2
Cube 1049	(Attempt 1052):	Rotations:	177,	Time:	0
Cube 1050	(Attempt 1053):	Rotations:	3663,	Time:	1
Cube 1051	(Attempt 1054):	Rotations:	511,	Time:	0
Cube 1052	(Attempt 1055):	Rotations:	3375,	Time:	1
Cube 1053	(Attempt 1056):	Rotations:	7395,	Time:	3
Cube 1054	(Attempt 1057):	Rotations:	497,	Time:	0

Command Prompt - python ai_learner.py -s 3 --seed=3

```
Correct: (Current: 5, Running: 9.922, Max: 28)
Reward: (Current: 7, Running: 13.906, Max: 35)
Randomness: 3.97%
Current Iter: 375000
Current Time: 223 seconds
```



```
Correct: (Current: 13, Running: 10.061, Max: 28)
Reward: (Current: 17, Running: 14.051, Max: 35)
Randomness: 3.97%
Current Iter: 376000
Current Time: 224 seconds
```

----- PAUSING -----

Attempt 7 (seed: 3)
Best cube (28 correct squares):



Really Quit? (y/n)>

28 squares

Command Prompt - python ai_learner.py -s 3 --seed=3

```
Correct: (Current: 11, Running: 11.999, Max: 32)
Reward: (Current: 14, Running: 16.336, Max: 40)
Randomness: 3.02%
Current Iter: 269000
Current Time: 450 seconds
```



```
Correct: (Current: 12, Running: 11.788, Max: 32)
Reward: (Current: 15, Running: 16.109, Max: 40)
Randomness: 3.02%
Current Iter: 270000
Current Time: 450 seconds
```

----- PAUSING -----

Attempt 14 (seed: 3)
Best cube (32 correct squares):



Really Quit? (y/n)>

32 squares

Command Prompt - python ai_learner.py -s 3 --seed=3

```
Correct: (Current: 7, Running: 7.412, Max: 36)
Reward: (Current: 10, Running: 10.798, Max: 44)
Randomness: 3.00%
Current Iter: 597000
Current Time: 281 seconds
```



```
Correct: (Current: 15, Running: 10.845, Max: 36)
Reward: (Current: 18, Running: 14.927, Max: 44)
Randomness: 3.00%
Current Iter: 598000
Current Time: 281 seconds
```

----- PAUSING -----

Attempt 29 (seed: 3)
Best cube (36 correct squares):



Really Quit? (y/n)> ^X

36 squares

SARSA Stats

```

[2 5]
[1 5]

[4 2]    [6 2]    [1 1]    [5 3]
[4 2]    [6 1]    [3 3]    [5 4]

[4 6]
[3 6]

[2 2]
[2 2]

[4 4]    [6 6]    [1 1]    [5 5]
[4 4]    [6 6]    [1 1]    [5 5]

[3 3]
[3 3]

***** SARSA LEARNING *****
=====

The goal state was reached in 11 episodes.
```

```

Goal state reached for episode 73
Action taken to reach s' front-face-right
Q value for (s, a) 9.504012637346158
Goal state reached
```

Deliverables



- ✓ Documentation Report (README.md)
- ✓ Python programming files (.py files) and the test results of the project.
- ✓ GitHub repository link
- ✓ YouTube video of the developed project

The background of the slide features a grid of numbers, likely from a Rubik's cube or a similar puzzle, with a black pen resting on it. The grid is partially obscured by a blue and purple gradient overlay on the left side.

Evaluation Methodologies

- ❖ The success of the project is determined by the successful implementation of the RL algorithms Q-Learning and SARSA.
- ❖ The agent must be able to successfully solve the cube to its initial position or must be able to solve most of the pieces.
- ❖ SARSA is preferable over Q-Learning to minimize errors which is shown from the above statistics.

Thank You

