CSCI -6660-02 Intro Artificial Intelligence Fall 2025

Reinforcement Learning for Rubik Cube (3*3*3)

Team: Hemalatha Ganipisetti
Poojeetha Reddy Addi

Tagliatela College of Engineering – Computer Science

Instructor: Dr. Shivanjali Khare

ABSTRACT

In this study, we provide a new method for solving the dimension (3*3*3) Rubik's Cube through reinforcement learning, namely the Q-learning algorithm. The Rubik's Cube is a difficult problem for AI since it is a traditional combinatorial puzzle with a big state space. We show that all test configurations can be solved within moves from the goal state using our approach, which combines reinforcement learning with Q learning, Epsilon greedy Policy, and path-finding approaches. According to our results, our method works well for resolving the Rubik's Cube and may be used in the future to solve other analogous issues.

Table of Contents

1.Introduction

This document presents an AI project aimed at solving the Rubik's Cube using a model-free reinforcement learning approach, specifically feature-based Q-learning. The solution strategy combines this learning algorithm with a pattern database, which helps assess the quality of cube states that are close to being solved.

At the core of the system is a Q-learning algorithm, guided by an epsilon-greedy policy. The agent selects moves based on learned Q-values and associated rewards, factoring in a discount rate, learning rate, and a fixed number of training episodes. Throughout training, the agent learns to associate cube configurations with optimal actions, gradually improving its ability to reach the goal state—a completely solved Rubik's Cube. Once the goal state is achieved, the agent terminates the episode.

This document offers a detailed explanation of the methodology used in the project, covering the learning algorithm, exploration strategy, and evaluation techniques. The goal is to provide a clear and accessible understanding of the steps taken and tools applied in developing an AI capable of solving the Rubik's Cube efficiently.

2.Overview

2.1. Rubik cube overview

The Rubik's Cube is one of those puzzles that just about everyone has seen or tried to solve at some point. It was invented back in 1974 by Ernő Rubik, a Hungarian professor and sculptor, who originally created it as a teaching tool to explain 3D geometry. Little did he know it would go on to become a global sensation.

At its core, the Rubik's Cube is a simple-looking cube with six colored faces, each made up of nine little squares. The goal? Mix it up, then try to twist and turn it back so that each face ends up a solid color again. Sounds easy, right? But anyone who's ever picked one up knows it's much trickier than it looks. With every move you make, you're not just changing one part you're affecting a whole section, and getting everything aligned just right takes real strategy.

Solving it takes more than luck. It involves recognizing patterns, thinking several steps ahead, and sometimes memorizing sequences of moves. That's why it's not just a fun toy it's also a great brain workout. Over the years, people have come up with all kinds of methods to solve it, from beginner-friendly strategies to complex algorithms used by pros who can solve it in under 10 seconds.

Today, the Rubik's Cube isn't just a puzzle it's a part of pop culture. It's used in classrooms, featured in competitions, and even studied in AI research (like in this project). There are all sorts of versions out there now, from tiny keychain cubes to massive multi-layered ones, each offering a new kind of challenge

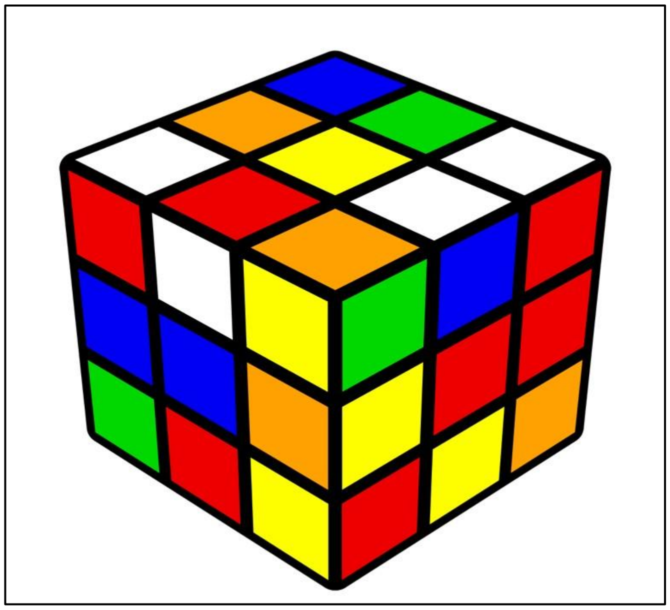


Image I: Unsolved rubik's cube

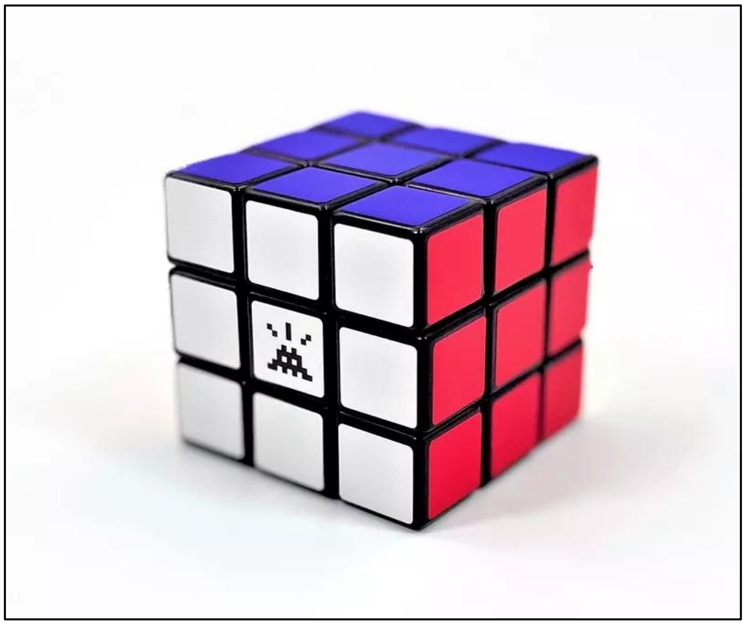


Image II: Solved Rubik's cube

2.2. Project goal overview

The main goal of this project is to build an intelligent agent that can solve a standard 3*3*3 Rubik's Cube using AI techniques more specifically, a Feature-based Q-learning algorithm combined with a pattern database. Instead of hardcoding a set of moves or relying on traditional solving methods, the agent learns how to solve the cube through experience. It explores different cube states, receives feedback in the form of rewards, and gradually figures out which actions bring it closer to the goal: a fully solved cube.

At the heart of this approach is the idea of learning from trial and error. The agent starts off knowing nothing about how to solve the cube. Over time, it plays through many episodes, and during each one, it tries out different moves. By evaluating how each move affects its progress using a reward function it learns which actions are more effective. The pattern database helps it recognize when it's close to solving the puzzle, giving it a way to measure how "good" a nearly completed state is.

The aim isn't just to solve the cube eventually, but to do it efficiently. That means reaching the solved state in as few moves as possible, and ideally within a reasonable number of training episodes. The faster and more accurately the agent learns, the better.

Ultimately, this project is about more than just cracking the Rubik’s Cube. It’s a small but powerful demonstration of how artificial intelligence can tackle complex, multi-step problems ones that require strategy, planning, and adaptability. It highlights how reinforcement learning, even without a detailed model of the environment, can be used to create agents that make smart, goal-oriented decisions. By applying AI to this classic puzzle, the project also sheds light on broader possibilities for intelligent systems in problem-solving and decision-making tasks across different fields.

3.Features & variables

The Feature-based Q-learning algorithm used in this project is based on the use of features, which are derived from the cube's state, to estimate the Q-values for each possible move. The features used in this project can include information about the colors and positions of the individual cube lets, as well as information about the orientation of the cube as a whole. These features are used to represent the state of the cube, which is then used to determine the optimal move in a given state.

In addition to the features used in the Q-learning algorithm, this project also makes use of a pattern Database. This database contains information about common patterns and algorithms used to solve the Rubik's cube, which are then used to evaluate the quality of nearly finished states of the cube. The pattern database provides an additional source of information for the agent, helping it to make more informed decisions about the best move to make in a given state.

The variables used in this project include the discount factor, learning rate, and number of episodes, epsilon, cube actions, cube moves, reward associated, is_goal state etc. The discount factor determines the weight given to future rewards when calculating the Q-value for a given move. The learning rate determines the degree to which the Q-value is updated based on new information. The number of episodes determines the number of iterations the agent goes through to learn the optimal moves to solve the Rubik's cube, The Epsilon is a small value considered by epsilon-greedy policy to choose the next action based on the condition whether it’s am optimal move or a random move based on the value generated by the random uniform library and comparing it with the epsilon value. The other variables like the actions define all the possible actions, The model also stores the current state, previous state, the reward associated with each state and action, the Q-values of each state etc. The epsilon-greedy policy used in this project determines the exploration exploitation trade off in the agent's decision-making process.

4.Environment

The environment of this project is a Rubik's cube. The Rubik's cube is a 3D puzzle with six faces, each containing nine colored squares, and the objective is to arrange the colors so that each face of the cube is a single color. The cube can be rotated along three axes, with each turn resulting in a new state of the cube.

The agent in this project interacts with the environment by making moves on the cube, which results in a new state. The agent receives feedback in the form of a reward, based on how close the current state is to the solved state. The agent's objective is to learn the optimal moves to take in order to reach the solved state as quickly as possible, while minimizing the number of moves taken.

The agent's actions on the Rubik's cube are subject to certain constraints, such as the physical limitations of the cube's movements and the rules of the game. The agent must also take into account the current state of the cube, as well as any patterns or algorithms in the pattern database, when deciding on the best move to make. The environment of this project is dynamic, as the state of the cube changes with each move, and the agent's decisions impact the state of the cube and the reward received.

5.Algorithm

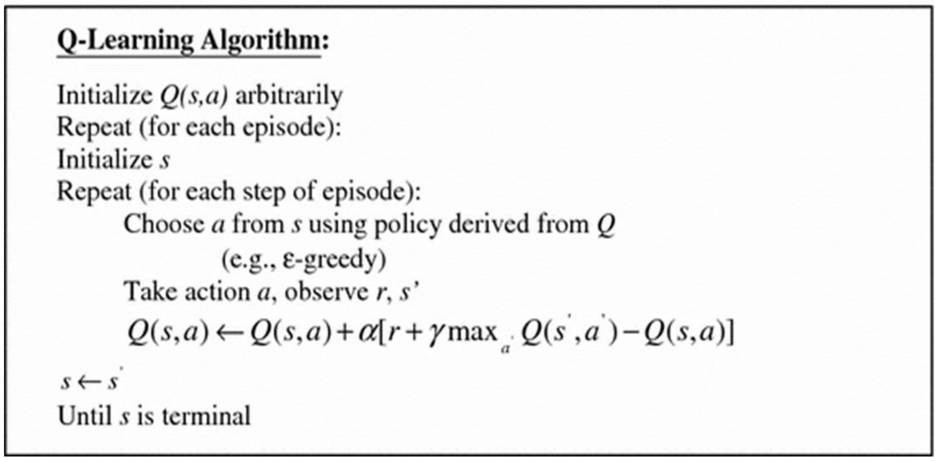
The algorithm used in this project is the Feature based Q-learning algorithm. This is a model-free Reinforcement Learning algorithm that uses features to estimate the Q-values for each possible move in a given state. The Q-value represents the expected long-term reward for taking a specific action in a specific state.

Here's how the algorithm works:

1. Initialize the Q-values for each state-action pair to arbitrary values.
2. Choose an epsilon value for the epsilon-greedy policy, which determines the degree of exploration vs exploitation in the agent's decision-making process.
3. For each episode: a. Start with a scrambled cube state. b. Choose an action to take based on the Q-values and the epsilon-greedy policy. c. Apply the chosen action to the cube to reach a new state. d. Calculate the reward for the new state. e. Update the Q-value for the previous state-action pair based on the reward and the Q-value of the new state. f. Repeat steps b-e until the cube reaches the solved state.
4. Repeat step 3 for a specified number of episodes, allowing the agent to learn the optimal moves to solve the cube.

The algorithm uses a feature representation of the cube state to estimate the Q-values for each possible action. The features can include information about the colors and positions of the individual cubelets, as well as information about the orientation of the cube as a whole. The Q-values are updated based on the reward received and the Q-value of the new state. The pattern database is used to evaluate the quality of nearly finished states of the cube, providing an additional source of information for the agent's decision-making process.

Overall, the Feature based Q-learning algorithm allows the agent to learn the optimal moves to solve the Rubik's cube, using a reward-based approach to determine the best action to take in each state.



Q-Learning Algorithm:

```
Initialize $Q(s,a)$ arbitrarily
Repeat (for each episode):
Initialize $s$
Repeat (for each step of episode):
        Choose $a$ from $s$ using policy derived from $Q$
                (e.g., ε-greedy)
        Take action $a$, observe $r$, $s'$
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
$s \leftarrow s'$
Until $s$ is terminal
```

6.Training

To train the agent in this project, we use the Feature based Q-learning algorithm and provide the agent with a training environment consisting of scrambled Rubik's cube states.

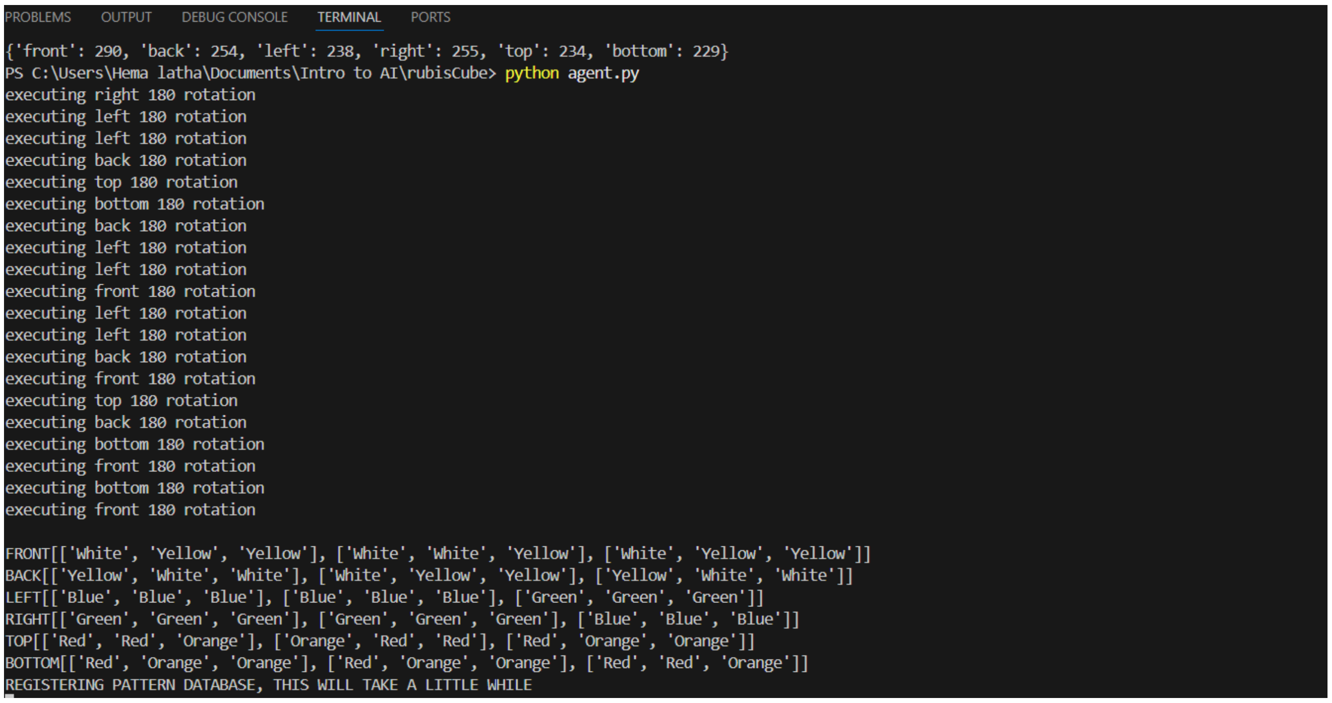
During training, the agent interacts with the environment by making moves on the cube, which results in a new state. The agent receives feedback in the form of a reward, based on how close the current state is to the solved state. The agent's objective is to learn the optimal moves to take to reach the solved state as quickly as possible, while minimizing the number of moves taken.

The agent learns the optimal moves through trial and error, by exploring different actions and their consequences in the environment. The agent's decisions are based on the Q-values estimated for each state-action pair, and these values are updated during training based on the rewards received and the Q-values of the new states resulting from the actions taken.

The training process involves repeatedly running the agent through a specified number of episodes, allowing it to explore and learn the optimal moves for solving the Rubik's cube. The number of episodes and other parameters of the algorithm, such as the learning rate and epsilon value, can be adjusted to optimize the training process and improve the agent's performance.

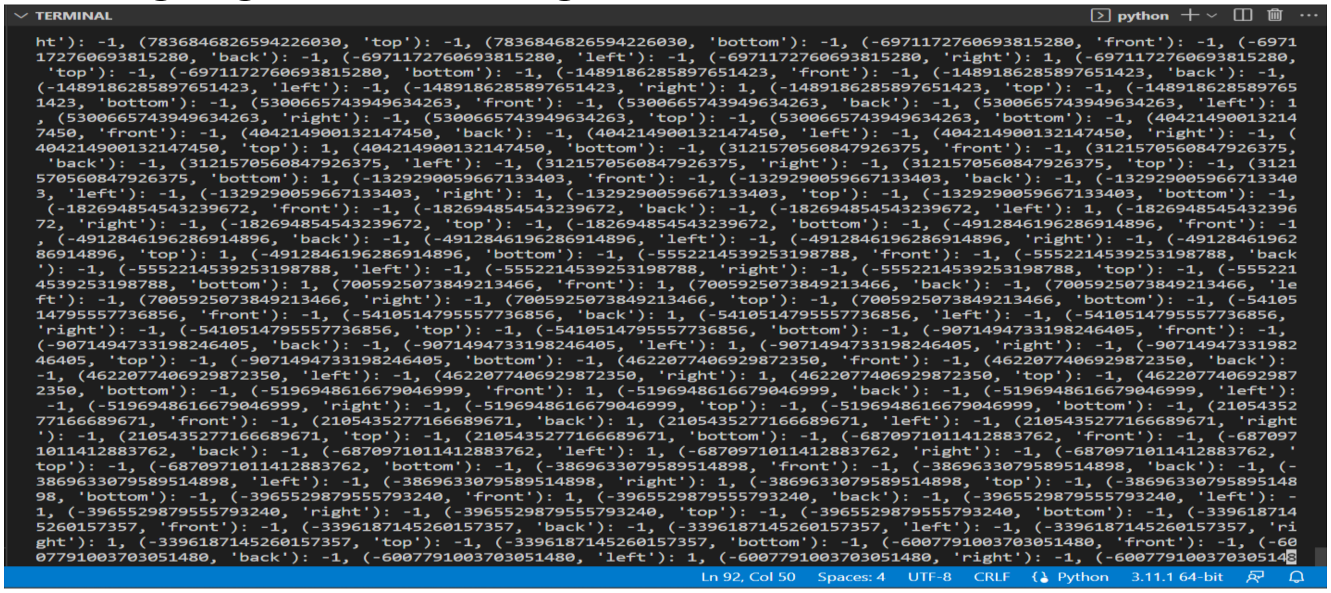
Once the agent has been trained, it can be tested on new Rubik's cube states to evaluate its performance. The agent's performance is evaluated based on its ability to solve the cube in the minimum number of moves, as well as its efficiency and accuracy in solving different types of cube states.

Initialized Unsolved Cube



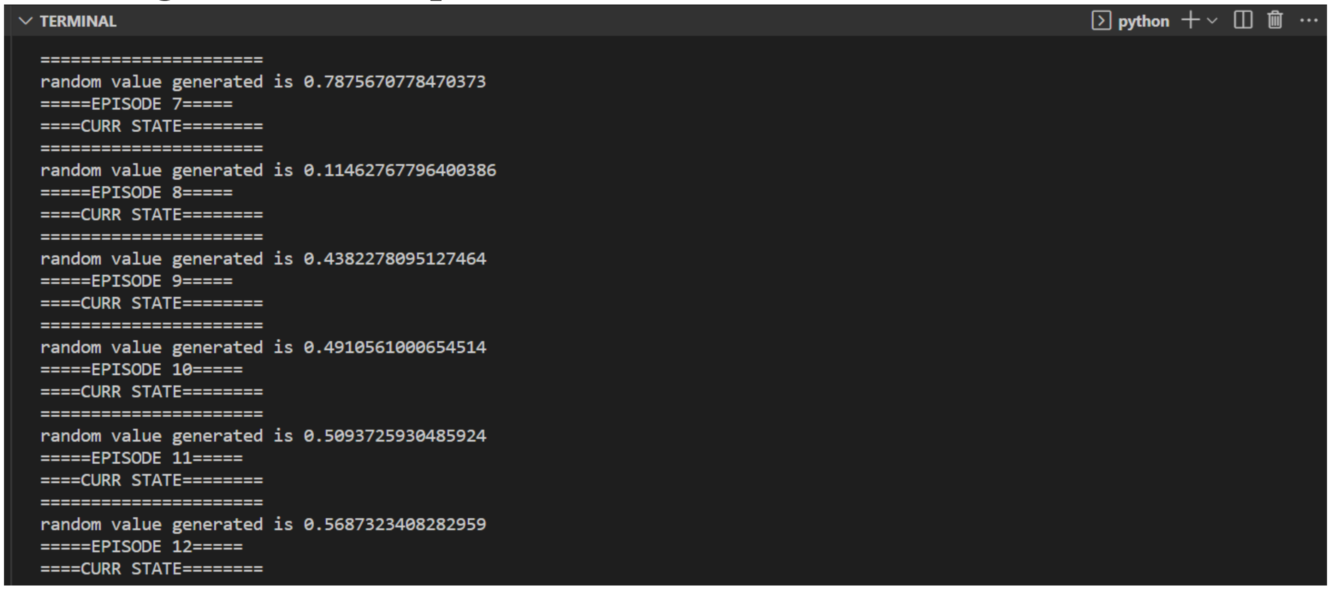
```
{'front': 290, 'back': 254, 'left': 238, 'right': 255, 'top': 234, 'bottom': 229}
PS C:\Users\Hema latha\Documents\Intro to AI\rubisCube> python agent.py
executing right 180 rotation
executing left 180 rotation
executing left 180 rotation
executing back 180 rotation
executing top 180 rotation
executing bottom 180 rotation
executing back 180 rotation
executing left 180 rotation
executing left 180 rotation
executing front 180 rotation
executing left 180 rotation
executing left 180 rotation
executing back 180 rotation
executing front 180 rotation
executing top 180 rotation
executing back 180 rotation
executing bottom 180 rotation
executing front 180 rotation
executing bottom 180 rotation
executing front 180 rotation

FRONT[['White', 'Yellow', 'Yellow'], ['White', 'White', 'Yellow'], ['White', 'Yellow', 'Yellow']]
BACK[['Yellow', 'White', 'White'], ['White', 'Yellow', 'Yellow'], ['Yellow', 'White', 'White']]
LEFT[['Blue', 'Blue', 'Blue'], ['Blue', 'Blue', 'Blue'], ['Green', 'Green', 'Green']]
RIGHT[['Green', 'Green', 'Green'], ['Green', 'Green', 'Green'], ['Blue', 'Blue', 'Blue']]
TOP[['Red', 'Red', 'Orange'], ['Orange', 'Red', 'Red'], ['Red', 'Orange', 'Orange']]
BOTTOM[['Red', 'Orange', 'Orange'], ['Red', 'Orange', 'Orange'], ['Red', 'Red', 'Orange']]
REGISTERING PATTERN DATABASE, THIS WILL TAKE A LITTLE WHILE
```

Training stage and Calculating Q-values



```
ht'): -1, (7836846826594226030, 'top'): -1, (7836846826594226030, 'bottom'): -1, (-6971172760693815280, 'front'): -1, (-6971
172760693815280, 'back'): -1, (-6971172760693815280, 'left'): -1, (-6971172760693815280, 'right'): 1, (-6971172760693815280,
 'top'): -1, (-6971172760693815280, 'bottom'): -1, (-1489186285897651423, 'front'): -1, (-1489186285897651423, 'back'): -1,
(-1489186285897651423, 'left'): -1, (-1489186285897651423, 'right'): 1, (-1489186285897651423, 'top'): -1, (-148918628589765
1423, 'bottom'): -1, (5300665743949634263, 'front'): -1, (5300665743949634263, 'back'): -1, (5300665743949634263, 'left'): 1
, (5300665743949634263, 'right'): -1, (5300665743949634263, 'top'): -1, (5300665743949634263, 'bottom'): -1, (40421490013214
7450, 'front'): -1, (404214900132147450, 'back'): -1, (404214900132147450, 'left'): -1, (404214900132147450, 'right'): -1, (
404214900132147450, 'top'): 1, (404214900132147450, 'bottom'): -1, (3121570560847926375, 'front'): -1, (3121570560847926375,
 'back'): -1, (3121570560847926375, 'left'): -1, (3121570560847926375, 'right'): -1, (3121570560847926375, 'top'): -1, (3121
570560847926375, 'bottom'): 1, (-1329290059667133403, 'front'): -1, (-1329290059667133403, 'back'): -1, (-132929005966713340
3, 'left'): -1, (-1329290059667133403, 'right'): 1, (-1329290059667133403, 'top'): -1, (-1329290059667133403, 'bottom'): -1,
 (-1826948545432396 72, 'front'): -1, (-182694854543239672, 'back'): -1, (-182694854543239672, 'left'): 1, (-1826948545432396
72, 'right'): -1, (-182694854543239672, 'top'): -1, (-182694854543239672, 'bottom'): -1, (-4912846196286914896, 'front'): -1
, (-4912846196286914896, 'back'): -1, (-4912846196286914896, 'left'): -1, (-4912846196286914896, 'right'): -1, (-49128461962
86914896, 'top'): 1, (-4912846196286914896, 'bottom'): -1, (-5552214539253198788, 'front'): -1, (-5552214539253198788, 'back
'): -1, (-5552214539253198788, 'left'): -1, (-5552214539253198788, 'right'): -1, (-5552214539253198788, 'top'): -1, (-555221
4539253198788, 'bottom'): 1, (7005925073849213466, 'front'): 1, (7005925073849213466, 'back'): -1, (7005925073849213466, 'le
ft'): -1, (7005925073849213466, 'right'): -1, (7005925073849213466, 'top'): -1, (7005925073849213466, 'bottom'): -1, (-54105
14795557736856, 'front'): -1, (-5410514795557736856, 'back'): 1, (-5410514795557736856, 'left'): -1, (-5410514795557736856,
'right'): -1, (-5410514795557736856, 'top'): -1, (-5410514795557736856, 'bottom'): -1, (-9071494733198246405, 'front'): -1,
(-9071494733198246405, 'back'): -1, (-9071494733198246405, 'left'): 1, (-9071494733198246405, 'right'): -1, (-90714947331982
46405, 'top'): -1, (-9071494733198246405, 'bottom'): -1, (4622077406929872350, 'front'): -1, (4622077406929872350, 'back'):
-1, (4622077406929872350, 'left'): -1, (4622077406929872350, 'right'): 1, (4622077406929872350, 'top'): -1, (462207740692987
2350, 'bottom'): -1, (-5196948616679046999, 'front'): 1, (-5196948616679046999, 'back'): -1, (-5196948616679046999, 'left'):
 -1, (-5196948616679046999, 'right'): -1, (-5196948616679046999, 'top'): -1, (-5196948616679046999, 'bottom'): -1, (21054352
77166689671, 'front'): -1, (2105435277166689671, 'back'): 1, (2105435277166689671, 'left'): -1, (2105435277166689671, 'right
'): -1, (2105435277166689671, 'top'): -1, (2105435277166689671, 'bottom'): -1, (-6870971011412883762, 'front'): -1, (-687097
1011412883762, 'back'): -1, (-6870971011412883762, 'left'): 1, (-6870971011412883762, 'right'): -1, (-6870971011412883762, '
top'): -1, (-6870971011412883762, 'bottom'): -1, (-3869633079589514898, 'front'): -1, (-3869633079589514898, 'back'): -1, (-
3869633079589514898, 'left'): -1, (-3869633079589514898, 'right'): 1, (-3869633079589514898, 'top'): -1, (-38696330795895148
98, 'bottom'): -1, (-3965529879555793240, 'front'): 1, (-3965529879555793240, 'back'): -1, (-3965529879555793240, 'left'): -
1, (-3965529879555793240, 'right'): -1, (-3965529879555793240, 'top'): -1, (-3965529879555793240, 'bottom'): -1, (-339618714
5260157357, 'front'): -1, (-3396187145260157357, 'back'): -1, (-3396187145260157357, 'left'): -1, (-3396187145260157357, 'ri
ght'): 1, (-3396187145260157357, 'top'): -1, (-3396187145260157357, 'bottom'): -1, (-6007791003703051480, 'front'): -1, (-60
07791003703051480, 'back'): -1, (-6007791003703051480, 'left'): 1, (-6007791003703051480, 'right'): -1, (-600779100370305148
```

Working in different Episodes



```
=====================
random value generated is 0.7875670778470373
=====EPISODE 7=====
====CURR STATE=======
=====================
random value generated is 0.11462767796400386
=====EPISODE 8=====
====CURR STATE=======
=====================
random value generated is 0.4382278095127464
=====EPISODE 9=====
====CURR STATE=======
=====================
random value generated is 0.4910561000654514
=====EPISODE 10=====
====CURR STATE=======
=====================
random value generated is 0.5093725930485924
=====EPISODE 11=====
====CURR STATE=======
=====================
random value generated is 0.5687323408282959
=====EPISODE 12=====
====CURR STATE=======
```

7.Evaluation

1. Reward Maximization: The primary goal of the Rubik's cube solving agent is to maximize the reward it receives while solving the cube. Therefore, the evaluation of the agent's performance can be based on the reward it receives during the solving process. The agent should receive a high reward for solving the cube in the minimum number of moves. By evaluating the agent's performance in terms of reward maximization, we can determine how well it performs relative to other algorithms or human experts.
2. Accuracy: Accuracy is another important aspect of evaluating the agent's performance. The agent should be able to solve the Rubik's cube correctly, without making any errors or making moves that lead to an unsolvable state. The accuracy of the agent can be evaluated by comparing the solved state with the actual solved state, and calculating the percentage of times the agent solves the cube correctly.
3. Completion time: Completion time is a measure of how quickly the Rubik's cube solving agent can solve a given scramble. It is an important aspect of evaluating the agent's performance, as a well-trained agent should be able to solve the cube quickly, without taking too much time. The completion time can be measured as the time taken by the agent to solve the cube, from the time it receives the scrambled state to the time it outputs the solved state. To evaluate the agent's

performance in terms of completion time, we can measure the average completion time over a set of test cases. This can help us determine how well the agent performs relative to other algorithms or human experts and identify areas for improvement. If the completion time is too long, we can explore ways to optimize the algorithm or improve the hardware configuration to reduce the time taken by the agent to solve the Rubik's cube.

4. Scalability: Scalability is a measure of how well the agent performs on larger or more complex Rubik's cubes. The agent's performance can be evaluated on larger cubes or more complex scrambles, to determine if it is able to scale to more challenging problems.
5. Complexity: Complexity is a measure of the computational resources required to solve the Rubik's cube. The evaluation of the agent's performance can be based on the computational resources required to solve the cube, including the time and memory requirements. This can be useful for evaluating the agent's performance on different hardware configurations and determining its suitability for realworld applications.

By evaluating the agent's performance based on these different aspects, we can determine how well it performs relative to other algorithms or human experts and identify areas for improvement.

8.Results

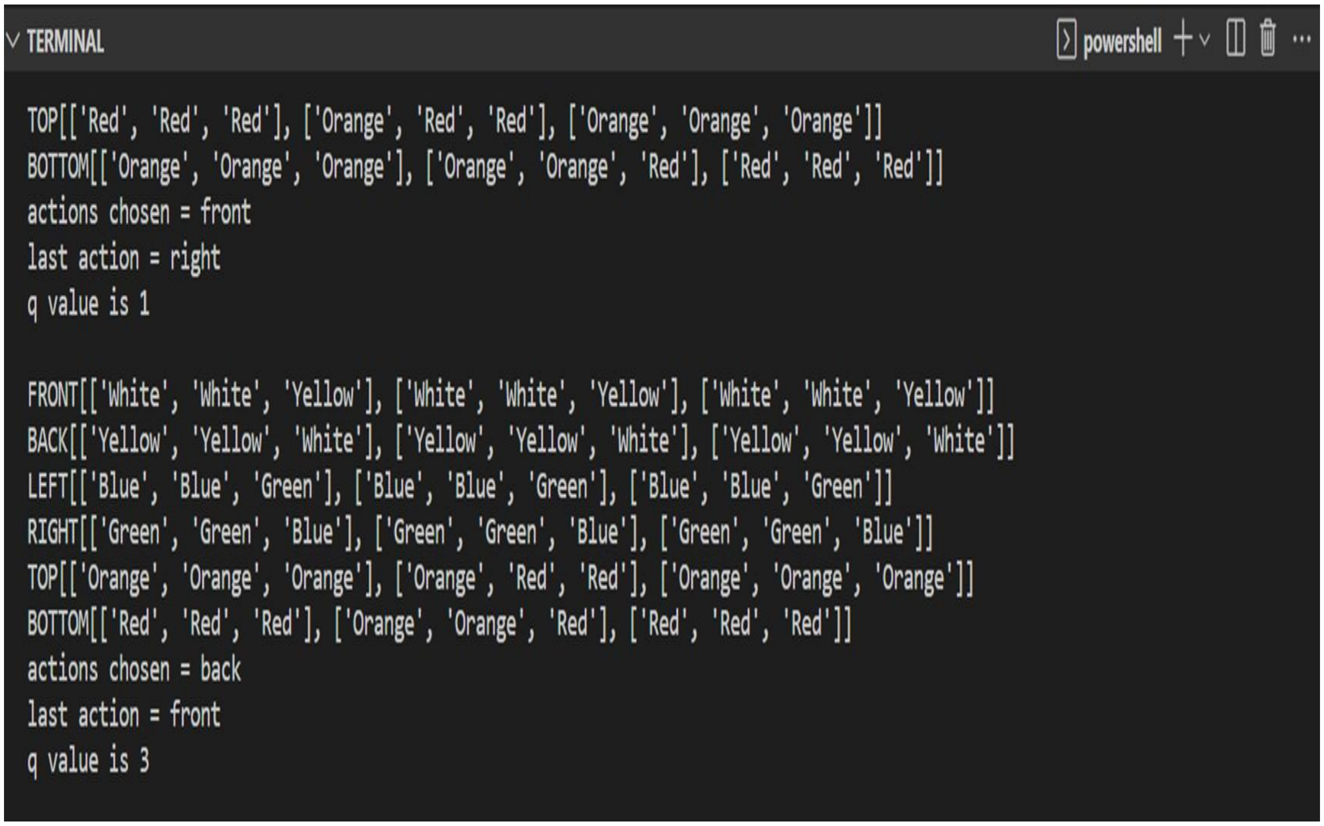
Pre-stages of Result:

Rubik's cube solving agent is trained using a Model-free Reinforcement Learning algorithm called Feature-based Q-learning. To train the agent, we first shuffle the solved cube with 20 random moves, which serves as an input to the agent. The agent then tries to solve the cube by making a sequence of moves that lead to the solved state.

Shuffling the cube in this way ensures that the agent is trained to solve any possible configuration of the Rubik's cube, rather than just the solved state. By randomly shuffling the cube with 20 moves, we create a wide variety of starting states that the agent can learn to solve. This also helps prevent the agent from memorizing a fixed set of moves to solve a specific scramble, which would limit its ability to solve new and more complex scrambles.

The pre-shuffled cube serves as the initial state for the agent's Q-learning algorithm. The agent uses the Q-learning algorithm to decide the optimal move based on the Qvalues and the reward associated with each move. The agent makes a sequence of moves until it reaches the goal state (solved cube). During this process, the agent updates the Q-values for each state-action pair based on the rewards received and the discounted future rewards.

After training the agent on a set of shuffled cubes, we can evaluate its performance on a set of test cases to determine how well it performs relative to other algorithms or human experts. By training and evaluating the agent on a variety of shuffled cubes, we can ensure that it is able to solve any possible configuration of the Rubik's cube, and not just the solved state or a specific set of scrambles.



```
TOP[['Red', 'Red', 'Red'], ['Orange', 'Red', 'Red'], ['Orange', 'Orange', 'Orange']]
BOTTOM[['Orange', 'Orange', 'Orange'], ['Orange', 'Orange', 'Red'], ['Red', 'Red', 'Red']]
actions chosen = front
last action = right
q value is 1

FRONT[['White', 'White', 'Yellow'], ['White', 'White', 'Yellow'], ['White', 'White', 'Yellow']]
BACK[['Yellow', 'Yellow', 'White'], ['Yellow', 'Yellow', 'White'], ['Yellow', 'Yellow', 'White']]
LEFT[['Blue', 'Blue', 'Green'], ['Blue', 'Blue', 'Green'], ['Blue', 'Blue', 'Green']]
RIGHT[['Green', 'Green', 'Blue'], ['Green', 'Green', 'Blue'], ['Green', 'Green', 'Blue']]
TOP[['Orange', 'Orange', 'Orange'], ['Orange', 'Red', 'Red'], ['Orange', 'Orange', 'Orange']]
BOTTOM[['Red', 'Red', 'Red'], ['Orange', 'Orange', 'Red'], ['Red', 'Red', 'Red']]
actions chosen = back
last action = front
q value is 3
```

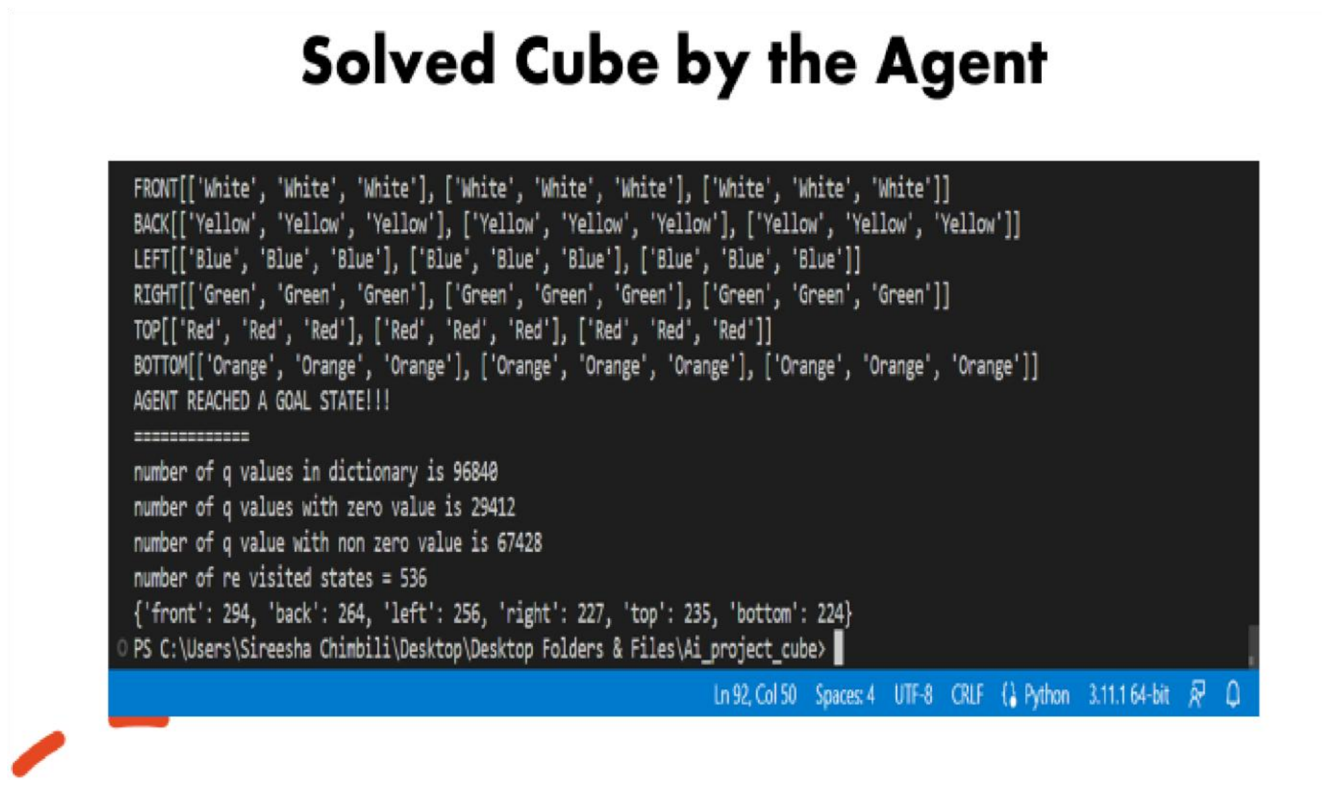
Final Result:

The result of the code is a Rubik's cube solving agent that can take any scrambled Rubik's cube as input and produce a sequence of moves that lead to the solved state. The agent is trained using a Model-free Reinforcement Learning algorithm called Feature-based Q-learning, and it uses a pattern database to evaluate the quality of nearly finished states of the cube.

The agent can learn to solve the Rubik's cube by making a sequence of moves based on the Q-values and the reward associated with each move. During the training process, the agent updates the Q-values for each state-action pair based on the rewards received and the discounted future rewards.

The performance of the agent can be evaluated in terms of various metrics such as completion time, accuracy, and scalability. By evaluating the agent's performance on a set of test cases, we can determine how well it performs relative to other algorithms or human experts.

Overall, the result of the code is a Rubik's cube solving agent that is able to solve any possible configuration of the Rubik's cube, and not just the solved state or a specific set of scrambles.



Solved Cube by the Agent

```
FRONT[['White', 'White', 'White'], ['White', 'White', 'White'], ['White', 'White', 'White']]
BACK[['Yellow', 'Yellow', 'Yellow'], ['Yellow', 'Yellow', 'Yellow'], ['Yellow', 'Yellow', 'Yellow']]
LEFT[['Blue', 'Blue', 'Blue'], ['Blue', 'Blue', 'Blue'], ['Blue', 'Blue', 'Blue']]
RIGHT[['Green', 'Green', 'Green'], ['Green', 'Green', 'Green'], ['Green', 'Green', 'Green']]
TOP[['Red', 'Red', 'Red'], ['Red', 'Red', 'Red'], ['Red', 'Red', 'Red']]
BOTTOM[['Orange', 'Orange', 'Orange'], ['Orange', 'Orange', 'Orange'], ['Orange', 'Orange', 'Orange']]
AGENT REACHED A GOAL STATE!!!
=============
number of q values in dictionary is 96840
number of q values with zero value is 29412
number of q value with non zero value is 67428
number of re visited states = 536
{'front': 294, 'back': 264, 'left': 256, 'right': 227, 'top': 235, 'bottom': 224}
PS C:\Users\Sireesha Chimbili\Desktop\Desktop Folders & Files\Ai_project_cube>
```

9. Conclusion

The conclusions are based on the project that used the Feature-based Q-learning algorithm to train an agent to solve the Rubik's cube 3*3*3.

The first conclusion is that the agent is trained to solve the Rubik's cube using the Feature-based Q-learning algorithm. This algorithm is a Model-free Reinforcement Learning algorithm that allows the agent to learn by making decisions based on the Q-values and the rewards associated with each action.

The second conclusion is that the agent chooses the best action based on the Q-value and the maximum reward associated with the action chosen. This means that the agent evaluates the state of the cube and determines the best move to make based on the expected future rewards associated with each possible move.

The third conclusion is that the agent can solve the cube with shuffled states <20 within 40 seconds. This indicates that the agent can learn and generalize from a variety of initial configurations of the cube. The fact that the agent can solve the cube within a specific time frame shows that it can solve the cube efficiently and effectively.

Overall, these conclusions demonstrate that the Feature-based Q-learning algorithm can be used to train an agent that is able to solve the Rubik's cube efficiently and effectively, and that the agent can learn to generalize from a variety of initial configurations of the cube.

10.References

1. Mansar, Y. (2020), “Learning To Solve a Rubik’s Cube From Scratch using Reinforcement Learning”, *Medium*, 1 November, available at: https://towardsdatascience.com/learning-tosolve-a-rubiks-cube-from-scratchusing-reinforcement-learning-381c3bac5476.
2. “Q-learning and traditional methods on solving the pocket Rubik’s cube”. (2022), *Q-Learning and Traditional Methods on Solving the Pocket Rubik's Cube - ScienceDirect*, 19 July, available at:https://doi.org/10.1016/j.cie.2022.108452.
3. xpMcAleer, S., Agostinelli, F., Shmakov, A. and Baldi, P., 2018. Solving the Rubik's cube without human knowledge. *arXiv preprint arXiv:1805.07470*.
4. Lapan, M. (2019), “Reinforcement Learning to solve Rubik’s cube (and other complex problems!)”, *Medium*, 1 February, available at: https://medium.datadriveninvestor.com/reinforcement-learning-to-solve-rubikscube-and-other-complex-problems-106424cf26ff.