Question 1

```
                tempArr.append(char)
            adjMatrix.append(tempArr)
            tempArr = []

        # run backtracking algorithm
        runMapColoring(states, colors, adjMatrix)

    #run script
    main()
```

```
Solution found:
NSW = B
NT = B
Q = G
SA = R
WA = G
V = G
T = Any color (Since Tasmania is not adjacent to any other state)
```

No two neighbouring states have the same color. For example, NT, SA and WA are neighbours, so none of them have the same color.

T is isolated, so it can have any color

And so on..

Sample output of question2

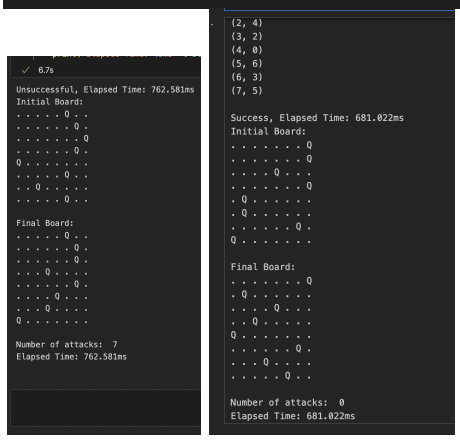
```
'T': 4000, 'decay': 0.99, 'epochs': 100000
```

```
Unsuccessful, Elapsed Time: 857.408ms
Initial Board:
. . Q . . . . .
. . . . Q . . .
. Q . . . . . .
. . Q . . . . .
. . . . . Q . .
. . Q . . . . .
. . . . Q . . .
. . . . . . Q .

Final Board:
Q . . . . . .
. . Q . . . . .
. . . . . Q . .
. . . . . Q . .
Q . . . . . .
. . Q . . . . .
. . . Q . . . .
. Q . . . . . .

Number of attacks:  6
Elapsed Time: 857.408ms
```

```
(2, 1)
(3, 6)
(4, 0)
(5, 3)
(6, 7)
(7, 4)

Success, Elapsed Time: 283.068ms
Initial Board:
Q . . . . . .
. . Q . . . . .
. . . . Q . . .
. . . . Q . . .
. . . . Q . . .
Q . . . . . .
. . . . . . Q
. . . . . Q .

Final Board:
. . Q . . . . .
. . . . . Q . .
. Q . . . . . .
. . . . . Q .
Q . . . . . .
. . . Q . . . .
. . . . . Q
. . . . Q . . .

Number of attacks:  0
Elapsed Time: 283.068ms
```

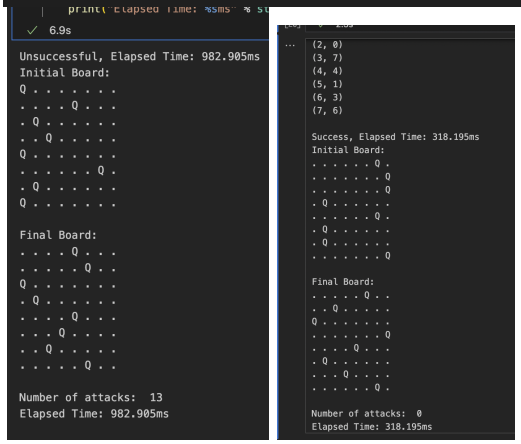
First right success, left unsuccessful

'T': 2000, 'decay': 0.99, 'epochs': 100000



Left unsuccess, right successful

'T': 1000, 'decay': 0.99, 'epochs': 100000



Left unsuccessful, right successful

Table

Settings	Average Time to Converge (ms)	Success Rate (probability)
Settings 1 (T = 4000)	607.556	0.41
Settings 2 (T = 2000)	663.576	0.47
Settings 3 (T = 1000)	611.275	0.49

From the observation, as the temperature decreases, the success rate increases. The time taken increases to 663 at T=2000 and decreases again. After 100s of tests, when k

decreases to 1000, it is more accurate compared to when T = 4000, and it will take a longer time.

Ran 100 times in each

```
if __name__ == '__main__':
    board = Board()
    param = {'T': 4000, 'decay': 0.99, 'epochs': 100000}

    time = 0
    turn = 0

    for i in range(100):
        sa = SimulatedAnnealing(board, param)
        elapsed_time, success = sa.run()
        # board.visualize()  # This will print the board state after the algorithm ends, regardless of success
        # print("Elapsed Time: %sms" % str(elapsed_time))
        time += elapsed_time

        if success:
            turn+=1


    print(time/100, success)
```

T = 4000,2000 and 1000

Ran a thousand times

```
            elapsed_time, see = sa.run()

            if see == True:
                success += 1
            #board.visualize()  # This will print the board state after the algorithm ends, regardless of success
            #print("Elapsed Time: %sms" % str(elapsed_time))
            totalTime += elapsed_time

        print(j, success, (totalTime / 100))
```

✓ 56m 38.8s

```
4000 405 4618.565059999997
2000 421 4554.706050000001
1000 431 4404.741530000003
```

+ Code + Markdown

Trend as temperature decrease, time decreases and success rate increases, but there will be a point when this will not happen and the success rate decrease instead since it doesnt explore too much.

Problem 3

```python
    # Policy representation for printing
    policy_repr = {0: '→', 1: '←', 2: '↓', 3: '↑'}

    # Generate policy
    policy = MDP_policy(S, A, P, U)

    # Print utilities and policy in a 4x4 grid
    print("Utilities and Policy for the Given Wumpus World:")
    for i in range(grid_size):
        for j in range(
            state = i *
            print(f"{U[state]:.2f} {policy_repr[policy[state]]}", end=" | ")
        print()
```

(variable) state: int

✓ 0.0s

```
Utilities and Policy for the Given Wumpus World:
6.63 → | 9.77 → | 14.12 → | 20.17 → |
4.37 → | 6.63 → | 9.77 → | 14.12 ↑ |
2.75 → | 4.37 ↑ | 2.03 → | 9.77 ↑ |
1.58 ↑ | −6.85 ↑ | −0.23 → | 6.63 ↑ |
```

When gamma = 0.9

```python
    # Policy representation for printing
    policy_repr = {0: '→', 1: '←', 2: '↓', 3: '↑'}

    # Generate policy
    policy = MDP_policy(S, A, P, U)

    # Print utilities and policy in a 4x4 grid
    print("Utilities and Policy for the Given Wumpus World:")
    for i in range(grid_size):
        for j in range(grid_size):
            state = i * grid_size + j
            print(f"{U[state]:.2f} {policy_repr[policy[state]]}", end=" | ")
        print()
```

8] ✓ 0.0s

```
Utilities and Policy for the Given Wumpus World:
0.73 → | 2.34 → | 5.72 → | 12.74 → |
−0.05 → | 0.73 → | 2.34 → | 5.72 ↑ |
−0.42 → | −0.05 ↑ | −3.87 → | 2.34 ↑ |
−0.43 ↑ | −10.02 ↑ | −4.65 → | 0.73 ↑ |
```

When gamma = 0.6

```python
    policy_repr = {0: '→', 1: '←', 2: '↓', 3: '↑'}

    # Generate policy
    policy = MDP_policy(S, A, P, U)

    # Print utilities and policy in a 4x4 grid
    print("Utilities and Policy for the Given Wumpus World:")
    for i in range(grid_size):
        for j in range(grid_size):
            state = i * grid_size + j
            print(f"{U[state]:.2f} {policy_repr[policy[state]]}", end=" | ")
        print()
```

2] ✓ 0.0s

```
Utilities and Policy for the Given Wumpus World:
−0.37 → | 0.11 → | 2.12 → | 10.51 → |
−0.49 → | −0.37 → | 0.11 → | 2.12 ↑ |
−0.50 ↓ | −0.49 ↑ | −4.97 → | 0.11 ↑ |
−0.41 ← | −10.10 ← | −5.09 → | −0.37 ↑ |
```

When gamma = 0.3