The A* search algorithm evaluates the nodes which are quite closer to the end node or goal in terms of the minimum overall cost which is calculated using a heuristic. A node's cost that needs to be evaluated comprises of three scores G score, H score and F score where G score is the absolute distance from the starting node to the current node, H score is the heuristic value i.e. estimated distance from the current node to the end node and the F score is the total score, and is the addition of the G and H scores. (Reference 3)

We use a priority queue to implement the open list which contains nodes that can be explored from the already explored nodes. The priority queue allows for repeated selection and removal of the minimum cost node from the open list, using the get method.
To implement the path cost and parent of each node, we used python dictionaries. Here in the Priority Queue, we can't check whether an element is in that set or not, So, we have used path cost to check if the node has been visited already.

The starting node is added to the open list, its path cost is assigned to zero, while its parent is assigned to None. While the open list is not empty, we get the highest priority i.e., lowest cost node in the open list and check if that node is the goal, if it is not, we find the six neighbours of that node and calculate the cost of the neighbours. If the neighbour has not been explored yet or there exists a better path in terms of lower cost, we add its cost to the path cost and mark its parent as the current node, and add the neighbour to the open list priority queue by calculating its f score After we find the goal, we work backwards in the parent dictionary to find the parent of each node in the path till we reach the start node. (Reference 2)

Assume n is the board size input. For the worst-case time complexity, it is O(n^2log(n^2)). For a bad quality heuristic that always returns a zero cost for every node, the algorithm transforms into that of Dijkstra's. We are using a priority queue so that the complexity for Dijkstra's is O(|E|log|V|), E is 6n^2 and V is n^2 so it becomes O(n^2log(n^2)). And A* , with a positive heuristic, will always explore fewer nodes than Dijkstra because Dijkstra will explore nodes with costs less than the path cost while A* will explore nodes with cost plus heuristic less than the past cost. So the worst-case time complexity is indeed O(n^2log(n^2)). As for the average case, we need to use a different perspective and the time complexity is O(6^(relative error * d)log(n^2)) with d being the length of the solution, the relative error is defined as = (h* – h) /h*, where h* is the actual cost of getting from the root to the goal, and h is the estimated cost from the heuristic.
6 being the branching factor. Same as the worst case, log(n^2) is the cost of using the priority queue. The average case time complexity will depend on the input. For example, if the direct paths from many low heuristic cost nodes to the end node are blocked, the error will be high and vice versa. It also depends on the length of the solution path. As for the space complexity, it will always be O(n^2). (Reference 1)

The heuristic used is path cost between two nodes assuming there are no obstacles, which is just the summation of the absolute values of the x and y differences of the coordinates, while if the product of x and y coordinate differences is negative, we return the minimum of those two differences after adding the absolute value of the summation of the differences. This heuristic was used as it never overestimates the number of steps to the goal, which we derived from a relaxed version of the problem when there were no obstacles in the grid.

It is an admissible heuristic as the cost it estimates to reach the goal is never higher than the actual lowest possible cost from the current point in the path, and so it never overestimates the cost to reach the goal. Its cost will always be less than or equal to real cost because there might be obstacles in the grid between the start node and the goal node. (Reference 4)

The cost of the heuristic function is constant O(1) while the overall search cost is exponential which calls the heuristic function each time and has a total cost of O(6^(relative error * d)log(n^2)) ,d being the length of the solution and 6 being the branching factor.

For the challenge, when the search problem is extended such that, we are allowed to use existing board pieces of a specific colour as part of our solution path at no cost. An optimal solution is now defined as a minimal subset of unoccupied cells that need to be 'captured' by this colour in order to form a continuous path from the start coordinate to the goal coordinate. We would first calculate the estimated cost to goal using the old heuristic function and then, for those blocks which are connected to other coloured blocks, we get the minimum estimated cost to goal among those connected coloured blocks and use this value as the new estimated cost to goal. The old heuristic function will not be admissible in this new problem because there might be cases where the actual cost is smaller than the cost calculated by the heuristic function.

References:

1.
Chapter 3, Section 5. Artificial intelligence : A modern approach, ebook, global edition : a modern approach. (2016). Pearson Education, Limited.

2.
Implementation of A* - Red Blob Games
https://www.redblobgames.com/pathfinding/a-star/implementation.html

3.
A* Path Finding Algorithm
https://medium.com/swlh/a-path-finding-algorithm-1dd35f49c164

4.
Admissible heuristic
https://en.wikipedia.org/wiki/Admissible_heuristic