Assignment 3
Problem Statement

The following algorithms have to be implemented and integrated with the animation platform as demonstrated in the class.

Note: Programs will assign nodes to one of the following categories, and rendering will be done automatically. The categories are – open, closed, start, goal, solution, kernel, boundary, deleted, relay, and old (including old open, old closed, rolled back - also used for piping).

(You can find colour coding in the documentation shared with you along with the framework)

Game algorithms:-

The user should be able to specify the depth and the branching factor of the game tree. Your algorithm should assign random values to the leaf nodes and as output, it should be able to display the path from the root node to the leaf node whose heuristic value is returned by the algorithm. Also, display the min-max value. To verify the correctness, compute the minimax value of the game tree using minimax algorithm and compare it with the one computed using your algorithm.

G1: Alpha-Beta- The user should be able to run Alpha-Beta from left to right as well as right to left. Compute minimax value, the subtree traversed and trace the path from where the minimax value comes.
[Animation - highlight the edges and the nodes visited by the algorithm]

G2: SSS*- Compute minimax value, the subtree traversed and trace the path from where the minimax value comes.
[Animation - highlight the edges and the nodes visited by the algorithm]

[Piping: User should be able to pipe SSS* after AlphaBeta to show that SSS* explores a subtree of what AlphaBeta does]

Search algorithms:-

[Piping: In general one should be able to pipe any algorithm after another one]

[Animation: In general the progress of the algorithm should be displayed with Open and Closed nodes. To end with path found being highlighted]

S1: A* and Weighted A*- Find and mark the least cost path on the graph generated by the platform. Display the cost of the solution. The user should be able to choose a different value of K and re-run the algorithm on the same problem overlaid, after assigning the previous closed, open and solution to “old” category. The new run should color the nodes as “open” and “closed” again and, find and mark the least cost path on the graph.

S2: DFID and Iterative Deepening A* (IDA*)- Show how they do a sequence of deepening searches. Color the open, closed, and the deepest nodes in each iteration as “boundary”. In the next iteration colour the nodes again. Show the final path found. Display the total number nodes seen and the cost of the path found. The user should be able to select the value of Delta and rerun on the same problem.

S3: Recursive Best First Search (RBFS)- Show the closed nodes and the open nodes, the rolled back nodes as “old”, the nodes in waiting as “relay”. Show the final path found. Display the number of nodes visited, and the cost of the final solution.

S4: Divide and Conquer Frontier Search (DCFS)- Mark the “tabu” nodes as “deleted”, and show “relay” and “open” nodes. Draw the backpointers to relay nodes. When the goal is found colour it blue, and also the relay node it is linked to. Color the other relay nodes as “deleted”. Do the same in recursive calls. Display the total nodes expanded and the number of nodes in the solution.

S5: Sparse Memory Graph Search (SMGS)- The user should be able to choose a value of N, the number of nodes that can be kept in memory. Mark the “boundary” nodes, the “relay nodes”, and the “deleted” nodes. Draw the backpointers to relay nodes. When the goal is found color it blue (solution path), and also the relay node it is linked to. Color the other relay nodes as “deleted”. Do the same in recursive calls. Display the total nodes expanded and the number of nodes in the solution.

S6: Beam Stack Search (BSS)- Show the closed nodes, the deleted nodes, and the open nodes. Show the first and the subsequent solutions found in white, and the final solution in blue. When the first solution is found enable the stop button, and if pressed colour the best solution in blue. Draw an inverted table to represent the stack, with f-min and f-max values.

S7: Breadth First Heuristic Search (BFHS), Divide and Conquer BFHS (DCBFHS), Beam Search using f-values, and Divide and Conquer Beam Search (DCBS).

BFHS. Colour the nodes “closed”, “open” and “solution found”.
DCBFHS and DCBS. Mark the boundary nodes, the relay nodes, open nodes, and the deleted nodes. When the goal is found colour the start, goal and relay node blue, and recursively do the animation.

S8: Divide and Conquer Beam Stack Search (DCBSS)- Show the relay nodes, with links to boundary nodes, and the open nodes. When backtracking, show the regeneration of the stack. After a goal has been found, show the recursive animation, mark the path, and enable the stop button. Show the final path in blue.
[4 bonus marks for good effort]

TSP:-

For the TSP animation, a random set of N nodes is the starting point. The user should be able to give the value of N. Assume that all the nodes are connected to each other and the cost of each edge is the Euclidean distance.

The best candidate solution should be displayed on the screen (shown in blue). The user should be able to choose how often the solution is displayed.

Generate a plot of the cost of solution vs time/iterations (wherever applicable).
At all times the cost of the current tour and best tour found so far should be displayed, as also the average cost of tours in a population-based method.

T1: TSP- Heuristic Greedy tour construction. The different constructive methods: Nearest neighbour, Greedy heuristic, Savings heuristic.The user should be able to try different algorithms on the same problem and compare results.

T2: TSP - Beam Search, Variable Neighborhood Descent and Tabu Search. A set of city exchange and edge exchange neighbourhood operators must be available to the user. The user can choose an algorithm, the beam width parameter, the tabu tenure and the number of iterations, and try out different algorithms on the same problem. The starting tour may be randomly generated or imported from the greedy approach.

T3: TSP - Simulated Annealing. The User should be able to choose the Temperature. Experiment with cooling rate, various neighbourhood functions. A set of city exchange and edge exchange neighbourhood operators must be available to the user. The starting tour may be randomly generated or imported from the greedy approach. Generate a plot of cost of solution and temperature vs. time

T4: TSP – Genetic Algorithm. User should be able to choose crossover functions, problem size, population size, replacement percentage, and display frequency. Display the current best (blue) tour, the population fitness, and the all-time best (black) tour found so far. Give a plot of average fitness and best fitness with time.

T5: TSP – Ant Colony Optimization. User should be able to choose alpha, beta parameters, evaporation rate, number of ants, and termination criterion. Design a display to show number of ants on an edge, and the amount of pheromone on edges. Show the best tour selected by an ant in every cycle (or every K cycles) [Alternative animation: Display edges with non-epsilon pheromone value with proportionate thickness after each cycle. Initially there may be more edges but later one or two tours may remain].