OVERALL DESCRIPTION: I solved all the tasks and for maximum points from the AutoGrader

QUESTIONS 1-4 I implemented BFS, DFS, UCS, A* through using two universal methods mainly.

universalSearch – implements GSA depending on the provided datastructure: Stack, Queue, PriorityQueue, PriorityQueueWithFunction **getChildren** – inserts elements to frontier, which is a algorithm specific datastructure set in **initializeSearch**. In the frontier i keep for each element the positional information, directional information, parent information, cost information and the heuristic value InitializeSearch gets its parameters from depthFirstSearch, breadthFirstSearch,uniformCostSearch, aStartSearch function.

Example execution: depthFirstSearch calls initializeSearch with stack. Initializesearch initiates the parameters and then calls univerSalsearch, which iteratively calls getChildren function. universalSearch returns the result to initializeSearch, which passes on the parameters to **getPath.**

https://github.com/AndresNamm/HKU_AI

I have included the algorithm execution graphics to my github page. File name is AlgorithmExecution.png

QUESTION 4.1

Cost/ Expanded	DFS	BFS	UCS	A*+ManH	A*+EucH
tinyMze	10/15	8/15	8/15	8/14	8/13
mediumMaze	130/146	68/269	68/269	68/221	68/226
bigMaze	210/390	210/620	210/620	210/549	210/557

QUESTION 5

For the Corners Problem in addition to state i defined a 4 element grid of Boolean values. This resulted in total of n * 2**4 state space.

QUESTION 6

I used a simple heuristic, which finds the shortest Manhattan distance through all the unvisited corners. I does this by choosing the closest corner to the current position, pops this corner from the list. Then it repeats seeking the closest corner to that corner and iteratively continues until the list is popped empty.

QUESTION 6.1

Q1: == ; Q2:non-trivial;Q3:admissible;Q4:>;Q5: consistent

QUESTION 7

Same Heuristic as in 6, however in here the positions for dots are used instead of corners. I actually considered many other heuristics, but finally chose this, because in a real life situation, when its not possible to calculate all the distances between all the nodes, this solution would have advantage compared to brute force calculation, because often brute force approach is not applicable.. It is also very clear and conscise and simple to understand for me later on.

QUESTION 8

getNumberOfAttacks

function getNumberOfAttacks depends on function checkDir, which checks the amount of queens in the direction to the righ. checkDir function depends on the checkLimits function, which does not allow the checkDir function to go out of bounds.

getBetterBoard

Test outs every possible replacement for queen in all the columns. Then calculates the cost for moving the queen. If its better than the current minimum, it stores this position as a replacement and updates the minimum value. If there is no cost improvement, it stores the position to the „equal“ array, which is the source of sideways moves in case there is no cost improving movement. The movement from „equal“ array is picked by random based on time.

modifying the stop criterion

In here I just added a counter variable, which gets incremented by one , when a sideways move is performed. It is set again to 0 if after n sideways moves there is again a cost improvemet. If the counter reaches to the limit and no placement for n queens is found, the program stops execution.