CSD 311:
ARTIFICIAL INTELLIGENCE

Project Title:

SOKOBAN

Group Title:

SOK'AI'

Group No:22

Kumar Makkapati	2010110357
Kolluru Jeshwanth	2010110351
Seeram Narasimha	2010110574
Anjani Nukala	2010110966
Pranav Inturi	2010110472

Table Of Contents:

Our Approach:

We used A* algorithm with the Manhatten Distance heuristic to solve the Sokoban Puzzles.

Walk Through Of The Program:

- The program starts by taking the input containing all the 6 test cases from the "testExamples.xsb" file.
- The program then separates each test case into XSB files and iteratively feeds them into the A*search() Function.
- We have also used the DFS algorithm in one of our test cases.
- As the agent moves,we check if it moves alone or it pushes a block along the way and have allocated a particular Letter for every move.
- When the agent moves alone we represent the move in lower case and when it moves with the block it is represented in upper case.Here, We check for a set of legal moves that the agent can make from all the possible moves(u,l,d,r,U,LD,R) from its current position.

Our Approach:

Non-Legal Moves:

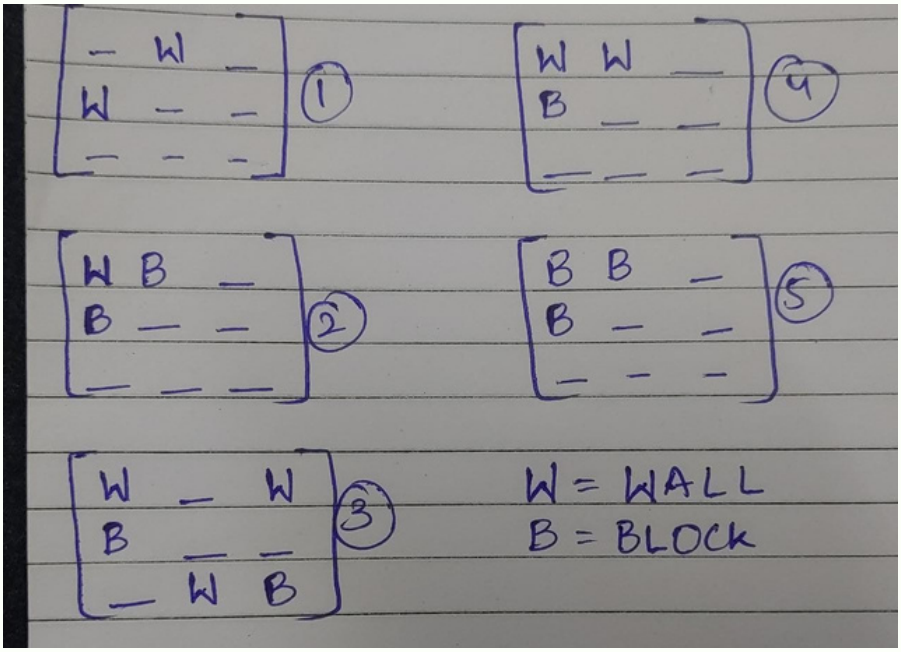
- The agent is being pushed into a position where there is a wall.
- The block that is being pushed by the agent is being moved into a position where there is another block or a wall.

- We calculated the manhattan distance value between the boxes and goals for each legal move and added the moves into a priority Queue with the heuristic value being the key.
- The moves are then executed in the order that they are popped from the priority queue.

- The output for each test case are appended into the file "output.xsb".Additional details like name of the test case, sequence of moves in each test case,number of moves and pushes made by the agent and time(in seconds) taken for each test case are printed in the terminal.

Handling Dead Lock:

- A level can become "deadlocked" as a result of the constraint of being able to push a box but never pull it, This indicates that no matter what the level is not solvable. Undoing a movement or restarting the level are the only options for completing the level.
- We have considered and handled a total of 5 Deadlock situations.
- The following picture shows the different deadlock situations in 3*3 size subgrids.



- For every legal move we have checked for these deadlock conditions and have skipped the moves which caused the deadlocks.

Other Algorithms and Heuristics Used:

- We first implemented the Euclidean distance Heuristic, but shortly after, we switched to the Manhattan distance as it worked better for higher dimensional data like this one.
- We also tried working on the Q-Learning algorithm as it is more efficient than A* but we couldn't perfectly implement it in the given time.

Future Scope:

- While researching about the project we came across algorithms like Min-Max algorithm,Hungarian Algorithm,Cart-Pole algorithm, they are surely more sophisticated and would yield better results in terms of time and number of moves and pushes.
- Our project doesn't allow backtracking,which can be very helpful if implemented as it decreases the time required for searching heavily.
- We weren't able to consider the situation where the blocks are already placed on goals.Due to this we weren't able to solve some sokoban puzzles of higher difficulty.
- More over,other than the standard algorithms we can also come up with our own creative solutions.For example,we can list out all the different paths from the boxes to the goals and check which ones of them are pushable by the agent.

References:

- https://www.lume.ufrgs.br/bitstream/handle/10183/190174/001088740.pdf?sequence=1#:~:text=Sokoban%20is%20a%20challenging%20state,cannot%20reach%20any%20goal%20state
- https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
- https://baldur.iti.kit.edu/theses/SokobanPortfolio.pdf