Sokoban Solver

Purva Jivani

Laxman Patel

Tanishq Gupta

Harshal Singh

This project implements an AI solver for Sokoban, a puzzle game where the player pushes boxes to target locations. The solver finds optimal sequences of moves for the player to push all boxes to goals without getting stuck.

Backend

Overview The backend is written in Python using heapq for priority queues and deque for BFS. It parses a level into

walls (#)
goals (.)
boxes ($)
player (@)
combined states (* and +).

Deadlock Detection

- Corner Deadlock: Box stuck in a corner not on a goal.
- Linear Deadlock: Box trapped along a wall without a goal in line.
- 2x2 Deadlock (optional): Two or more boxes trapped in a 2x2 square.
- Two-Box Freeze: Two boxes blocking each other along walls without goals.

Heuristic Function

Player Pathfinding Uses BFS to find paths for the player to reach positions necessary to push boxes. BFS avoids walls and boxes, ensuring valid movement.

Hungarian Algorithm: Computes minimum total Manhattan distance between boxes and goals.
If Hungarian fails, uses simple sum of nearest Manhattan distances.

A Search Algorithm*

- State = (boxes_positions, player_position)
- g(n) = cost so far (number of moves)

- h(n) = heuristic estimate (**Hungarian distance**)
  Priority = f(n) = g(n) + h(n)

- Expands nodes while avoiding deadlocks.
- Uses caching for BFS paths.
- Tracks came_from to reconstruct the move sequence.

Frontend

- The GUI is built using **Tkinter**, providing an interactive Sokoban interface with a clean layout.
- **Levels** can be selected from a list, loaded dynamically, and reset as needed.
- Supports **manual moves** (arrow keys/buttons), **undo**, **step-by-step solver**, and **auto-play** of the solution.
- Uses **pngs** for : PLAYER_PNG, BOX_PNG, BOX_GOAL_PNG, WALL_PNG, and TARGET_PNG.
- Smooth **animations** show player and box movements with interpolation for visual effect.
- **Move tracking** displays current moves vs total solution moves, and a pop-up appears upon level completion.

Results

Solver outputs number of expansions, total moves, and full move sequence. Handles moderately complex levels efficiently with optimized heuristics.

Conclusion

The Sokoban solver combines **A*** search with **BFS** and the **Hungarian heuristic** to efficiently find optimal or near-optimal solutions. A* explores possible game states, while BFS computes valid player movements to push boxes. The Hungarian heuristic estimates the minimum total distance of boxes to goals, guiding the search effectively. **Deadlock detection** (corner, linear, 2×2, and two-box freeze patterns) **prunes** unsolvable states. This combination ensures both efficiency and accuracy, solving complex Sokoban levels with minimal moves.