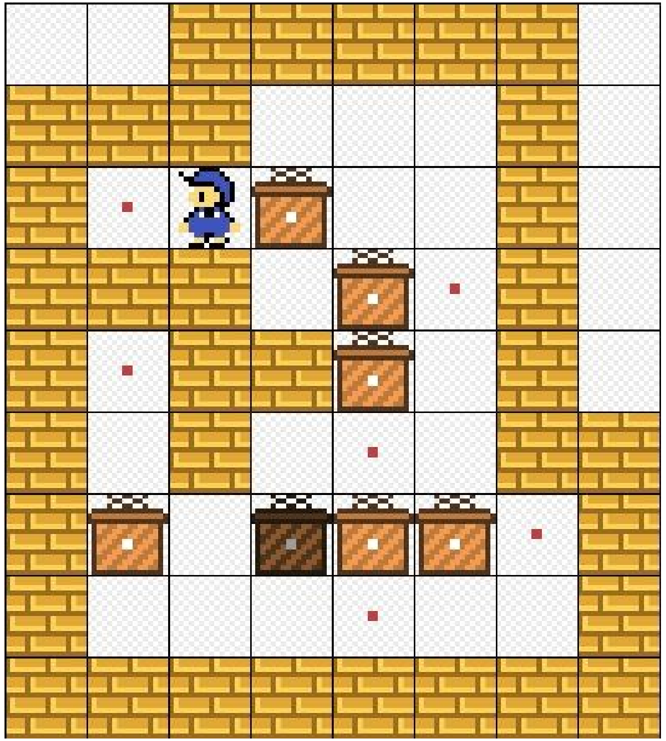
Implementation of IDA* for Sokoban

Joseph Nash, Anton Vasick, and Faraz Zaerpoor

Sokoban

- 2-dimensional grid
- Player, walls, boxes, and storage locations
- Player can make moves in any of the cardinal directions
- Moving into a box pushed the box in same direction
- Object of the puzzle is to move such that all boxes are occupying storage locations



Iterative Deepening A*

- Any node *n* has a value *f(n)* = *g(n)* + *h(n)*
    - *g(n)* = cost
    - *h(n)* = the heuristic function
- Runs branch and bound from a starting node
    - Halts when the node's *f* value exceeds a given cost
- If no goal node is found at this cost depth, the maximum cost is incremented and the branch and bound is run again

Iterative Deepening A*

- Any node *n* has a value $f(n) = g(n) + h(n)$
  - $g(n)$ = cost
  - $h(n)$ = the heuristic function
- Runs branch and bound from a starting node
  - Halts when the node's *f* value exceeds a given cost
- If no goal node is found at this cost depth:
  - Maximum cost is incremented to next contour (one more move)
  - Run again

IDA* Properties

- If *h* admissible, IDA* renders optimal path
- Time Complexity: $O(b^d)$
    - Same as A*
- Space Complexity: $O(bd)$
    - Same as depth-first search

Heuristics

- Naive
  - How many boxes are on switches
- Modified Naive
  - How many boxes are on switches
  - How far away is the guy from the closest box
- Modified Gale-Shapley
  - Create preference lists for the boxes and the switches
  - Run Gale-Shapley algorithm to create a stable matching
  - Find the distance from a box to its paired switch

Heuristics

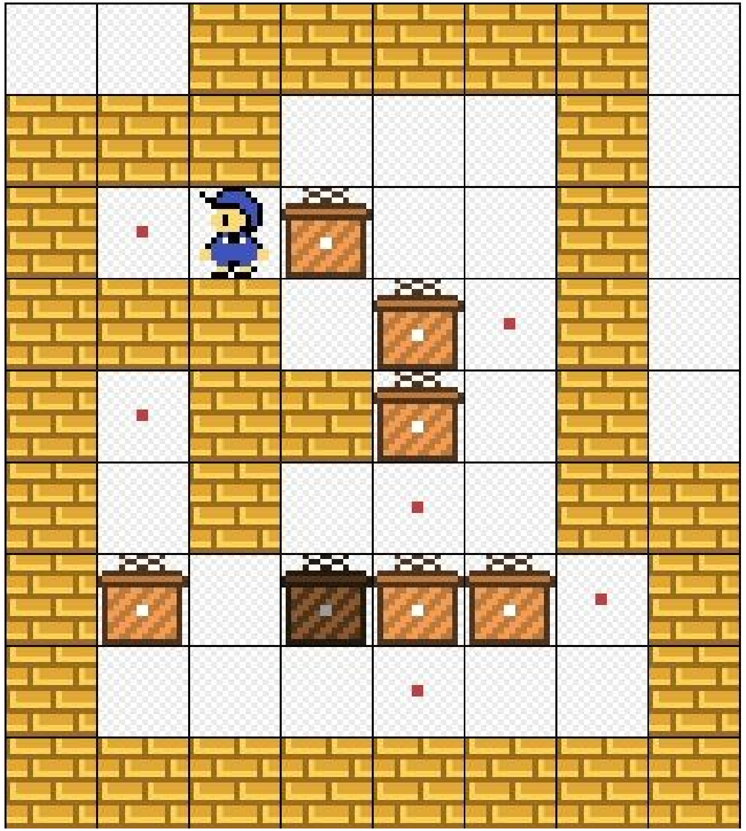
- Closest Switch
  - For each box, find the distance to the closest switch.
  - 2 boxes can be closest to the same switch
- MinMatching
  - Find the mapping of boxes to switches with the shortest number of moves
  - Solver use
  - Bootstrapping heuristic
  - Stochastics

Other Optimizations

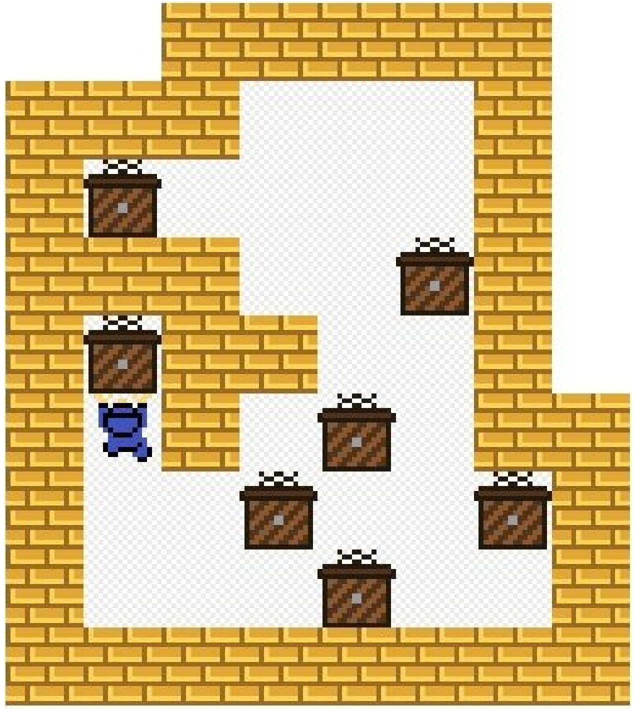
- Repeated State Elimination
  - Hash table of seen states
    - Hash function: label all squares, xor occupied lables, modulo prime
  - If a state has already been seen, do not add it to the open list
- Deadlock Avoidance
  - Sokoban can have unsolvable states
  - Detect these states and do not expand them
  - We found 1 box DL only, there are multi-box DL, i.e.
    - ####
    -   $$

Results

8x9, 7 box Sokoban puzzle:



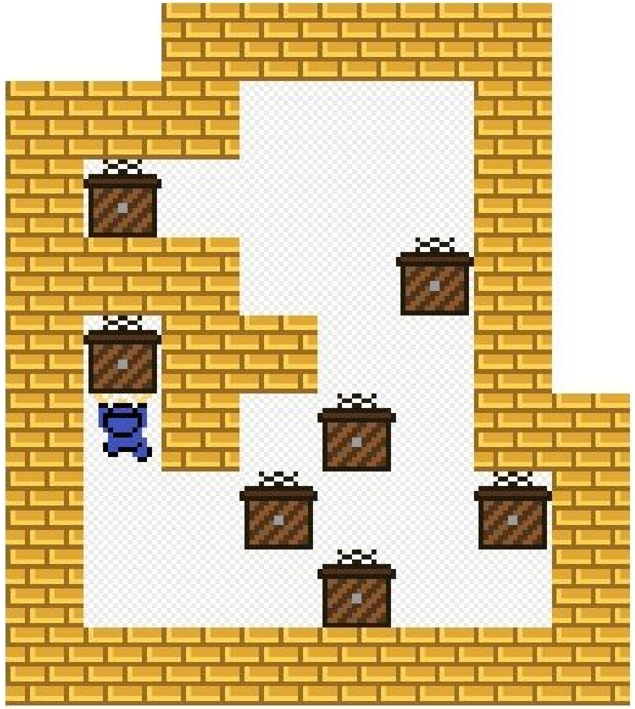
Results (cont)



```
['R','U','R','R','D','D','D','D','L','D',
'R','U','U','U','U','L','L','L','R','D','
R','D','R','D','D','L','L','D','L','L','U
','U','D','R']
```

- 34 steps
- 49.4 seconds

Results (cont)



Board Name	Precomputation	Total Runtime
sokoban0.txt	0.533839	0.60842 sec
sokoban1.txt	35.29168	dnr (>1 hour)
sokoban2.txt	dnr	dnr
sokoban3.txt	dnr	dnr
sokoban4.txt	dnr	dnr
sokoban5.txt	dnr	dnr
wikisoko.txt	19.84415	113.6* sec

- 49.4, modified naieve

Conclusions

- Helpful features
  - Good heuristic
  - Hashing
  - Dead square precomputation
- Needed features
  - Board shape macros (i.e. tunnel)
  - Advanced deadlock detection