Ricochet Robots

Solver Algorithms

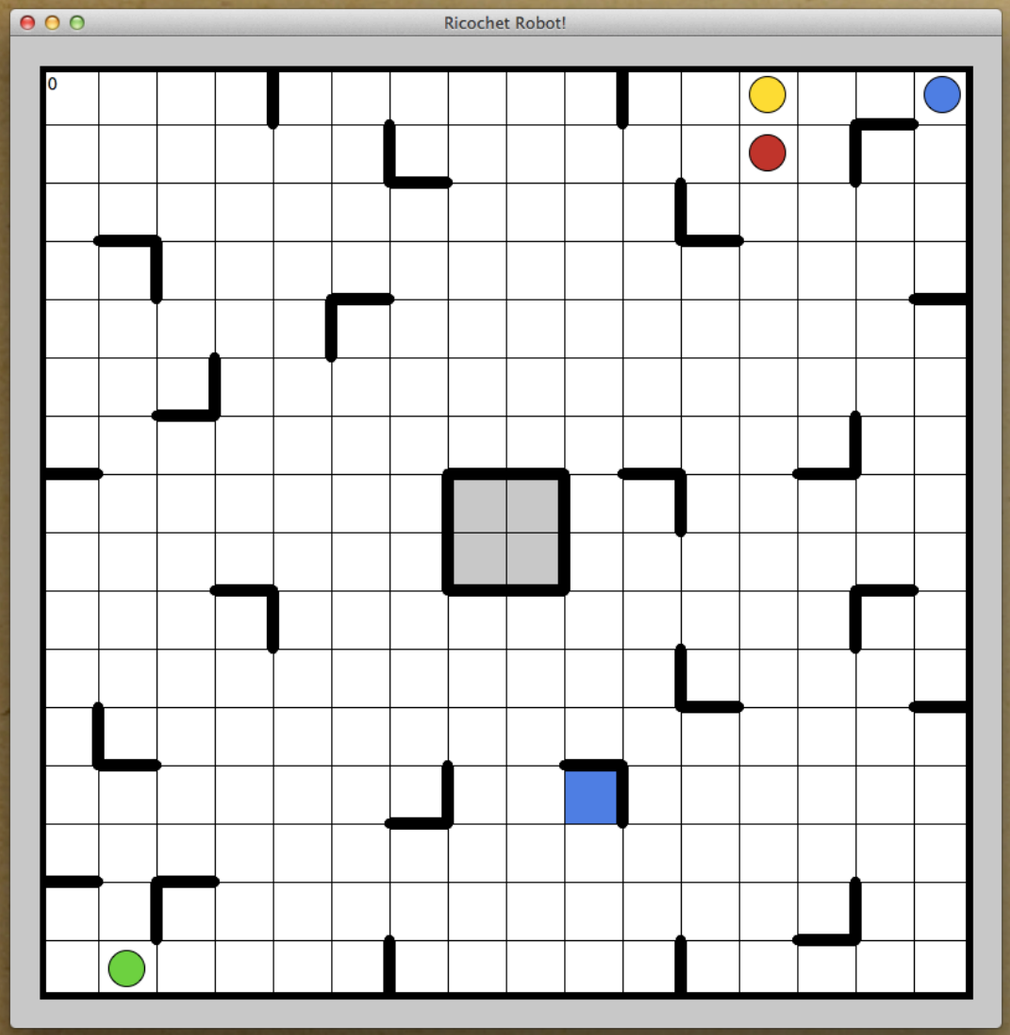
by Michael Fogleman

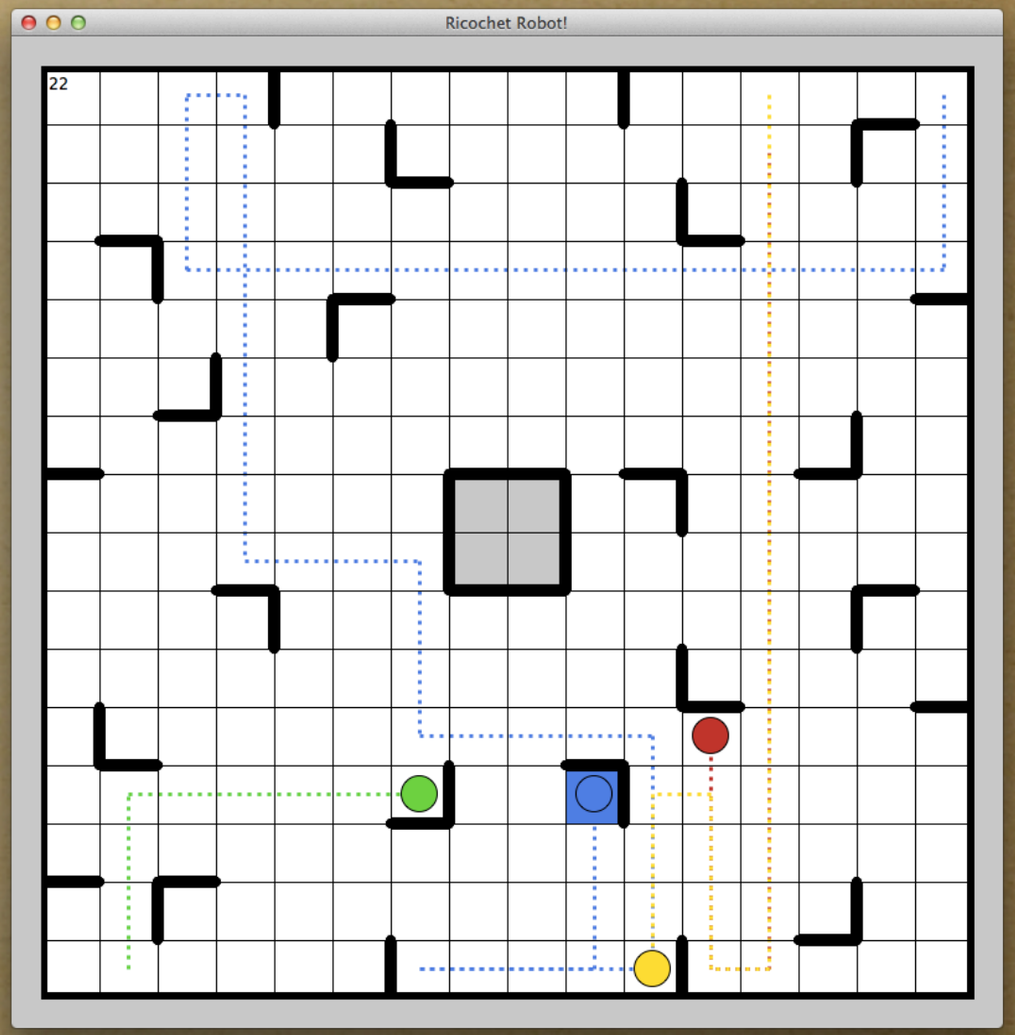
November 2012

It’s a board game!

Rules

- **16** x **16** board with **4** robots
- Robots move like rooks in chess, but cannot stop until they hit an obstacle (a wall or another robot)
- **Goal**: Get the "active" robot to the "target" square in the fewest moves possible
- Use the non-active robots as helpful obstacles





22-Move Example

Data Structures

- Array of **256** integers (**16** x **16** - one for each square) with **5** bits used for each
- One bit for each direction (**N**, **S**, **E**, **W**) to represent presence of a wall in that direction
- One bit to represent presence of a robot
- Array of **4** integers to represent position of each robot (so we don't have to scan for them)

Search

- Use iterative deepening, depth-first search
- Start with max search depth = 1, perform the search, increment max depth and repeat until a solution is found
- Guaranteed to find an optimal solution

Brute Force?

- Brute-force is untenable for even moderately complex puzzles
- 4 robots and 4 directions... maximum branching factor of **16**
- Searching just **10** moves deep requires evaluating **2,294,544,866** positions!
- It would take years to compute a **22** move solution

Memoization?

- Keep track of searched positions as we go
- If current position has already been searched to an equal or greater depth, prune the search
- This helps significantly, as duplicate positions are very common in the search tree

Hash Table

- **256** squares and **4** robots means we can represent a position with a single **32**-bit integer!
- Implement a hash table mapping **32**-bit integer (position) to another integer (depth searched)
- **22**-move puzzle becomes solvable with about **3** minutes of processing time

Improvement

- When generating the **32**-bit position key, sort the positions of the non-active robots first, as they can be transposed with no consequence
- Result: **22**-move puzzle can be solved in around **1** minute!

Pre-computing Stuff?

- Pre-compute the **minimum** number of moves needed to reach the target square from all other squares (assuming the robot could stop moving at anytime, like a rook, e.g. if other robots were in ideal positions)

- While searching, prune search whenever active robot cannot reach the target square in the number of moves remaining for the current search depth

Pre-computed Map

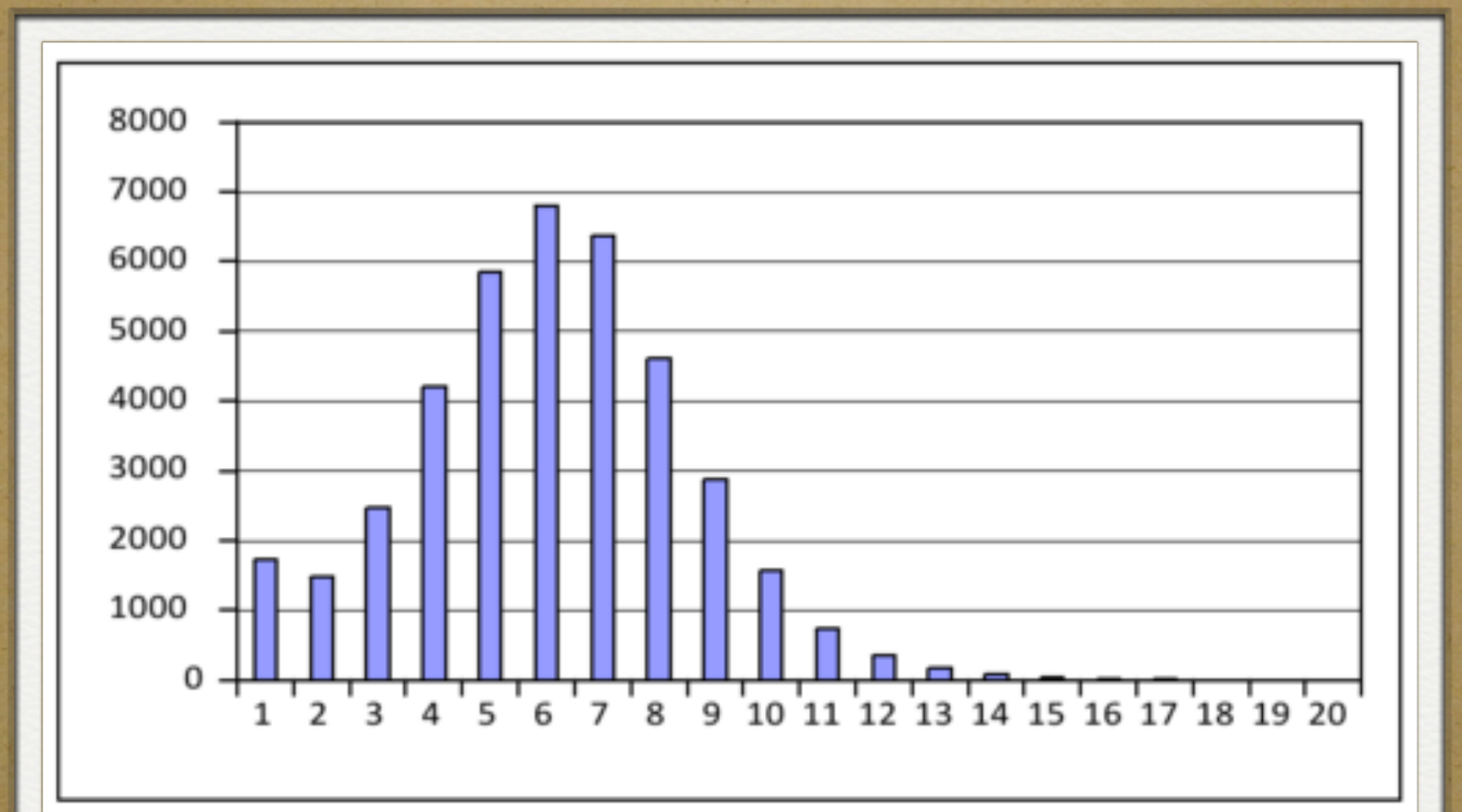
5	5	5	5	3	4	4	4	4	4	4	4	3	4	4	4
4	4	4	4	3	4	4	4	4	4	4	4	3	4	5	5
4	4	4	4	3	4	4	4	4	4	4	4	3	4	4	4
5	4	4	4	3	4	4	4	4	4	4	4	3	4	4	4
4	4	4	4	3	3	4	4	4	4	4	4	3	4	4	4
5	4	5	4	3	3	4	4	4	4	4	4	3	4	4	4
4	4	3	4	3	3	4	4	4	4	4	4	3	4	5	5
3	4	3	4	3	3	4			4	3	4	3	3	4	4
3	4	3	4	3	3	4			4	3	4	3	3	4	4
3	4	3	3	3	3	3	2	2	3	3	3	3	3	3	4
3	3	3	3	3	3	3	2	2	3	3	4	3	3	3	4
3	3	3	3	3	3	3	2	2	3	3	3	3	3	3	3
3	3	3	3	3	3	4	1	1	**X**	3	3	3	3	3	3
2	2	2	2	2	2	2	2	2	1	2	2	2	2	2	2
4	3	2	2	2	2	2	2	2	1	2	2	2	2	3	3
4	3	3	3	3	3	2	2	2	1	2	3	3	4	3	3

Results

- Searching to depth **10** now involves looking at just **34,269** positions
- Searching **22** moves deep covers around **47,612,050** positions
- Solution is found in about **10 seconds**!

Solver Output

```
Depth: 1, Nodes: 1 (0 inner, 0 hits)
Depth: 2, Nodes: 1 (0 inner, 0 hits)
Depth: 3, Nodes: 1 (0 inner, 0 hits)
Depth: 4, Nodes: 3 (1 inner, 0 hits)
Depth: 5, Nodes: 28 (11 inner, 0 hits)
Depth: 6, Nodes: 200 (100 inner, 34 hits)
Depth: 7, Nodes: 984 (577 inner, 293 hits)
Depth: 8, Nodes: 3779 (2431 inner, 1455 hits)
Depth: 9, Nodes: 12162 (8295 inner, 5429 hits)
Depth: 10, Nodes: 34269 (24349 inner, 16854 hits)
Depth: 11, Nodes: 87105 (63928 inner, 45994 hits)
Depth: 12, Nodes: 204091 (154241 inner, 114306 hits)
Depth: 13, Nodes: 446518 (346590 inner, 263199 hits)
Depth: 14, Nodes: 919338 (729935 inner, 565531 hits)
Depth: 15, Nodes: 1796912 (1451686 inner, 1142404 hits)
Depth: 16, Nodes: 3367854 (2758316 inner, 2197992 hits)
Depth: 17, Nodes: 6091594 (5050855 inner, 4069434 hits)
Depth: 18, Nodes: 10663998 (8946467 inner, 7281399 hits)
Depth: 19, Nodes: 18096372 (15356593 inner, 12617875 hits)
Depth: 20, Nodes: 29794218 (25564989 inner, 21196014 hits)
Depth: 21, Nodes: 47612050 (41300695 inner, 34539268 hits)
Depth: 22, Nodes: 1124068 (953336 inner, 793116 hits)
BS, BW, BN, BE, BS, BE, GN, GE, BS, RS, RW, RN, BE, BS, BW,
YS, YW, YN, YW, YS, BE, BN
```



Distribution of Moves Required

Most puzzles can be solved in under 10 moves

Get the Code

- git clone https://github.com/fogleman/Ricochet.git
- ./build_ricochet (requires gcc)
- python main.py (requires Python 2.x and wxPython)

Controls

- **R**, **G**, **B**, **Y**: Select robot by color
- **Arrows**: Move selected robot
- **N**: New Game
- **U**: Undo
- **S**: Solve

That's all!

- Contact: michael.fogleman@gmail.com
- I challenge you to implement a faster solver!