



# Sliding blocks puzzle solver

---

Mike Clancy  
U.C. Berkeley  
CS Division



## The assignment

---

- Write a program to solve sliding blocks puzzles.
- Example: Take over the human part of <http://www.puzzleworld.org/SlidingBlockPuzzles/pennant.htm>





## Framework of “try” procedure

---

- If current configuration is the goal, then return success; if current configuration has been seen before, then return failure.
- Register current configuration as seen.
- For each possible move, call “try” with the configuration that results from making that move:
  - If success, return success.
- Return failure.



## Grading

---

- Solution is run on easy puzzles.
- Iff it solves them, it is run on hard puzzles.
- Total points =
  - if easy puzzles solved, then score for easy puzzles + score for hard puzzles
  - else score for easy puzzles only.



## Use

---

- End-of-term project in U.C. Berkeley CS 2 (handed out four weeks prior to due date)
  - Most solutions are ~1000 lines of Java code.
- Easily configurable for less ambitious courses (even a CS 1 with backtracking search)



## Niftiness (1)

---

- Accommodates fast computers
- Encourages incremental development and modular design (make it work *correctly* before making it work *efficiently*)
- Has a large solution space; some efficiency constraints conflict with others
- Provides challenge for hotshots
- Is accompanied by lots of infrastructure
- Can be straightforwardly tweaked to counter possible cheating



## Niftiness (2)

---

- Students like it!