Game playing

- Types of games
  - Deterministic vs. chance
  - Perfect vs. imperfect information

- Active area of research
  - Why?
    - Clear criteria for success
    - Interesting, hard problems
    - Fun

- Typical game
  - 2-player, zero sum game
  - Players alternate moves
  - Perfect information, no chance
  - Examples?

Backtracking, minimax, game search

- We’ll use tic-tac-toe to illustrate the idea, but it’s a silly game to show the power of the method
  - What games might be better? Problems?

- Minimax idea: two players, one maximizes score, the other minimizes score, search complete/partial game tree for best possible move
  - In tic-tac-toe we can search until the end-of-the-game, but this isn’t possible in general, why not?
  - Use static board evaluation functions instead of searching all the way until the game ends

- Minimax leads to alpha-beta search, then to other rules and heuristics

Computer v. Human in Games

- Computers can explore a large search space of moves quickly
  - How many moves possible in chess, for example?

- Computers cannot explore every move (why) so must use heuristics
  - Rules of thumb about position, strategy, board evaluation
  - Try a move, undo it and try another, track the best move

- What do humans do well in these games? What about computers?
  - What about at Duke?

Minimax for tic-tac-toe (see ttt.cpp)

- Players alternate, one might be computer, one human (or two computer players)

- Simple rules: win scores +10, loss scores −10, tie is zero
  - X maximizes, O minimizes

- Assume opponent plays smart
  - What happens otherwise?

- As game tree is explored is there redundant search?
  - What can we do about this?
Backtracking/Mini-max from tt.cpp

```cpp
int Game::bestMove(Board::Player p, int & move)
  { // check for game over or too deep in search first
    int best = (p == Board::X ? COMPUTER_WIN : HUMAN_WIN);
    int score;
    int dontCareMove;
    for (k=0; k < myBoard.size(); k++) {
      if (myBoard.isClear(k)) { // can we move here?
        myBoard.place(k,p);
        score = bestMove(opposite(p),dontCareMove);
        myBoard.unplace(k);
        if (scoreIsBetter(score, best,p)) {
          best = score;
          move = k;
        }  
      }
    }
    return best;  
  }
```

Caching or Memoization

- In Tic-Tac-Toe do we see the same board more than once?
  ```
  X O .        X ? .
  X ? .        X O .
  . . .        . . .
  ```

- Repercussions in terms of search tree?
  - Does avoiding search result in significant savings?
  - How can we easily do this? Hint: maps!

- Lessons applied more widely
  - More storage results in lower runtime, general tradeoff
  - Can we have too much of a good thing?

Heuristics

- Can do pruning - see alpha-beta
- World will still be too big
  - Checkers: \(\sim10^{40}\) states
  - Chess: \(\sim10^{120}\) states

- A heuristic is a rule of thumb, doesn’t always work, isn’t guaranteed to work, but useful in many/most cases
  - Search problems that are “big” often can be approximated or solved with the right heuristics

Real-world performance

- Checkers: Chinook
- Chess: Deep Blue
- Othello: TD-gammon
- Go - not so good