Motion as Search

- Motion planning as path-finding problem
  - Problem: configuration space is continuous
  - Problem: under-constrained motion
  - Problem: configuration space can be complex

Approximate Decomposition

- Break configuration space into a grid
  - Search (A*, etc)
  - What can go wrong?
  - If no path found, can subdivide and repeat
- Problems?
  - Still scales poorly
  - "Incomplete"
  - Wiggly paths

Hierarchical Decomposition

- Actually used in practical systems
- But:
  - Not optimal
  - Not complete
  - Still hopeless above a small number of dimensions

Announcements

- Project 1.2 is up (Single-Agent Pacman)
  - Critical update: make sure you have the most recent version!
- Reminder: you are allowed to work with a partner!
- Change to John’s section: M 3-4pm now in 4 Evans

Why are there two paths from 1 to 2?
Skeletonization Methods

- Decomposition methods turn configuration space into a grid
- Skeletonization methods turn it into a set of points, with preset linear paths between them

Visibility Graphs

- Shortest paths:
  - No obstacles: straight line
  - Otherwise: will go from vertex to vertex
  - Fairly obvious, but somewhat awkward to prove
- Visibility methods:
  - All free vertex-to-vertex lines (visibility graph)
  - Search using, e.g. A*
  - Can be done in $O(n^3)$ easily, $O(n^2 \log(n))$ less easily
- Problems:
  - Bang, screech!
  - Not robust to control errors
  - Wrong kind of optimality?

Voronoi Decomposition

- Voronoi regions: points colored by closest obstacle
- Voronoi diagram: borders between regions
  - Can be calculated efficiently for points (and polygons) in 2D
  - In higher dimensions, some approximation methods

Voronoi Decomposition

- Algorithm:
  - Compute the Voronoi diagram of the configuration space
  - Compute shortest path (line) from start to closest point on Voronoi diagram
  - Compute shortest path (line) from goal to closest point on Voronoi diagram.
  - Compute shortest path from start to goal along Voronoi diagram
- Problems:
  - Hard over 2D, hard with complex obstacles
  - Can do weird things:

Probabilistic Roadmaps

- Idea: just pick random points as nodes in a visibility graph
- This gives probabilistic roadmaps
  - Very successful in practice
  - Lets you add points where you need them
  - If insufficient points, incomplete, or weird paths

Roadmap Example
Potential Field Methods

- So far: implicit preference for short paths
- Rational agent should balance distance with risk!
- Idea: introduce cost for being close to an obstacle
- Can do this with discrete methods (how?)
- Usually most natural with continuous methods

Potential Fields

- Cost for:
  - Being far from goal
  - Being near an obstacle
  - Go downhill
  - What could go wrong?

Adversarial Search

[DEMO 1]

Game Playing State-of-the-Art

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.
- Chess: Deep Blue defeated human world champion Gary Kasparov in a six-game match in 1997. Deep Blue examined 200 million positions per second, used very sophisticated evaluation and undisclosed methods for extending some lines of search up to 40 ply.
- Othello: human champions refuse to compete against computers, which are too good.
- Go: human champions refuse to compete against computers, which are too bad. In go, b > 300, so most programs use pattern knowledge bases to suggest plausible moves.
- Pacman: unknown

Game Playing

- Axes:
  - Deterministic or stochastic?
  - One, two or more players?
  - Perfect information (can you see the state)?

- Want algorithms for calculating a strategy (policy) which recommends a move in each state

Deterministic Single-Player?

- Deterministic, single player, perfect information:
  - Know the rules
  - Know what actions do
  - Know when you win
  - E.g. Freecell, 8-Puzzle, Rubik’s cube

- ...it’s just search!
- Slight reinterpretation:
  - Each node stores the best outcome it can reach
  - This is the maximal outcome of its children
  - Note that we don’t store path sums as before
  - After search, can pick move that leads to best node
Deterministic Two-Player

- E.g. tic-tac-toe, chess, checkers
- Minimax search
  - A state-space search tree
  - Players alternate
  - Each layer, or ply, consists of a round of moves
  - Choose move to position with highest minimax value = best achievable utility against best play
- Zero-sum games
  - One player maximizes result
  - The other minimizes result

Minimax Example

Minimax Search

Minimax Properties

- Optimal against a perfect player. Otherwise?
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - $O(bm)$
- For chess, $b \approx 35$, $m \approx 100$
  - Exact solution is completely infeasible
  - But, do we need to explore the whole tree? [DEMO2: minHeur]

Resource Limits

- Cannot search to leaves
- Limited search
  - Instead, search a limited depth of the tree
  - Replace terminal utilities with an eval function for non-terminal positions
- Guarantee of optimal play is gone
- More plies makes a BIG difference
  - [DEMO 3: limitedDepth]
- Example:
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - $\alpha - \beta$ reaches about depth 8 – decent chess program
Evaluation Functions

- Function which scores non-terminals
- Ideal function: returns the utility of the position
- In practice: typically weighted linear sum of features:
  \[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]
- e.g. \( f_1(s) = \text{num white queens} - \text{num black queens} \), etc.

Evaluation for Pacman

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

Iterative Deepening

- Iterative deepening uses DFS as a subroutine:
  1. Do a DFS which only searches for paths of length 1 or less. (DFS gives up on any path of length 2)
  2. If “1” failed, do a DFS which only searches paths of length 2 or less.
  3. If “2” failed, do a DFS which only searches paths of length 3 or less.
  …and so on.

- This works for single-agent search as well!
- Why do we want to do this for multiplayer games?

α-β Pruning Example

α-β Pruning

- General configuration
  - \( \alpha \) is the best value the Player can get at any choice point along the current path
  - If \( n \) is worse than \( \alpha \), MAX will avoid it, so prune \( n \)'s branch
  - Define \( \beta \) similarly for MIN

α-β Pruning Pseudocode
**α-β Pruning Properties**
- Pruning has no effect on final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering":
  - Time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth
  - Full search of, e.g. chess, is still hopeless!
- A simple example of metareasoning, here reasoning about which computations are relevant

**Non-Zero-Sum Games**
- Similar to minimax:
  - Utilities are now tuples
  - Each player maximizes their own entry at each node
  - Propagate (or back up) nodes from children

**Stochastic Single-Player**
- What if we don’t know what the result of an action will be? E.g.,
  - In solitaire, shuffle is unknown
  - In minesweeper, don’t know where the mines are
- Can do expectimax search
  - Chance nodes, like actions except the environment controls the action chosen
  - Calculate utility for each node
  - Max nodes as in search
  - Chance nodes take average (expectation) of value of children
- Later, we’ll learn how to formalize this as a Markov Decision Process

**Stochastic Two-Player**
- E.g. backgammon
- Expectiminimax (!)
  - Environment is an extra player that moves after each agent
  - Chance nodes take expectations, otherwise like minimax

**Stochastic Two-Player**
- Dice rolls increase $b$: 21 possible rolls with 2 dice
  - Backgammon ≈ 20 legal moves
  - Depth 4 = $20 \times (21 \times 20)^4$ $1.2 \times 10^8$
  - As depth increases, probability of reaching a given node shrinks
  - So value of lookahead is diminished
  - So limiting depth is less damaging
  - But pruning is less possible...
  - TDGammon uses depth-2 search + very good eval function + reinforcement learning: world-champion level play

**What’s Next?**
- Make sure you know what:
  - Probabilities are.
  - Expectations are.
- Next:
  - How to learn evaluation functions
  - Markov Decision Processes