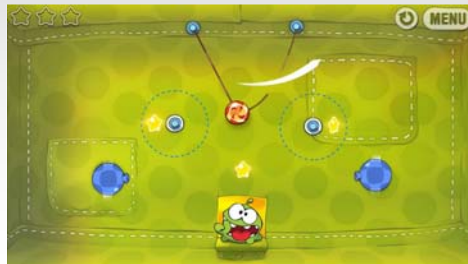
CPSC 436D
Video Game Programming

Strategy &
Adversarial Strategy



© Alla Sheffer

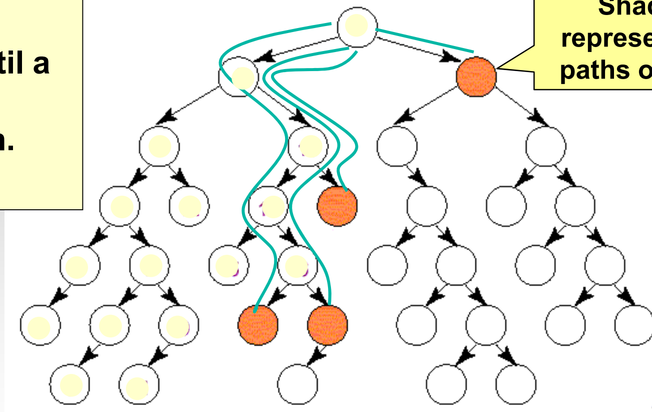
Strategy

- Given current state, determine **BEST** next move
- Short term: best among immediate options
- Long term: what brings something closest to a goal
  - *How?*
    - Search behavior tree for path to best outcome

© Alla Sheffer

DFS: Depth First Search

Explore each path on the frontier until its end (or until a goal is found) before considering any other path.

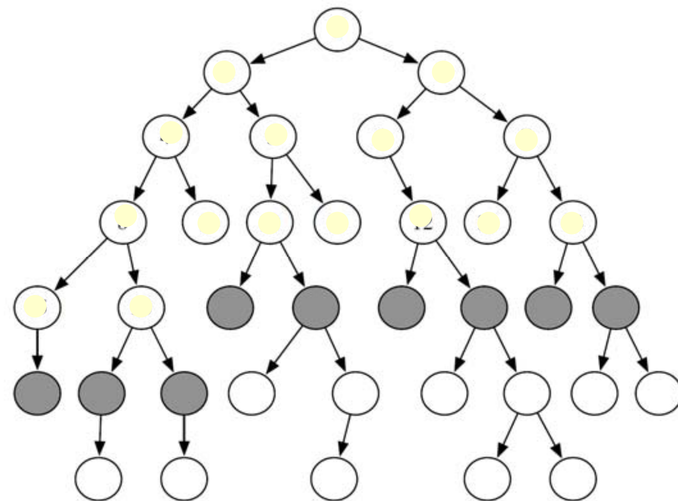


Shaded nodes represent the end of paths on the frontier

© Alla Sheffer

Breadth-first search (BFS)

- Explore all paths of length *l* on the frontier, before looking at path of length *l + 1*



© Alla Sheffer

When to use BFS vs. DFS?

- *The search graph has cycles or is infinite*

  BFS
- *We need the shortest path to a solution*

  BFS
- *There are only solutions at great depth*

  DFS
- *There are some solutions at shallow depth*

  BFS
- *No way the search graph will fit into memory*

  DFS

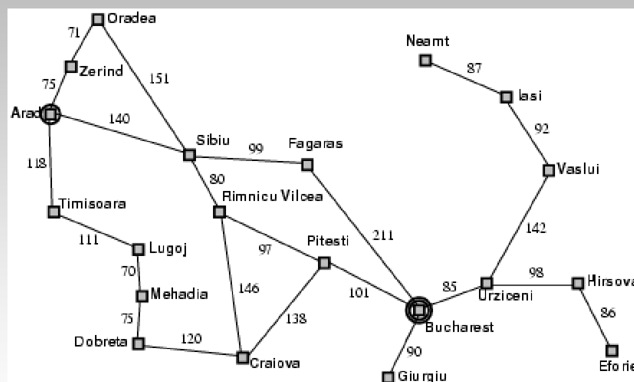
© Alla Sheffer

Search with Costs

Def.: The cost of a path is the sum of the costs of its arcs

$$\text{cost}(\langle n_0, \ldots, n_k \rangle) = \sum_{i=1}^{k} \text{cost}(\langle n_{i-1}, n_i \rangle)$$

Want to find the solution that minimizes cost



© Alla Sheffer

Copyright: Alla Sheffer

Lowest-Cost-First Search (LCFS)

- **Lowest-cost-first search finds the path with the lowest cost to a goal node**
- **At each stage, it selects the path with the lowest cost on the frontier.**
- **The frontier is implemented as a priority queue ordered by path cost.**

7

© Alla Sheffer

Use of search

- Use search to determine next state (next state on shortest path to goal/best outcome)
- Measures:
  - *Evaluate goal/best outcome*
  - *Evaluate distance (shortest path in what metric?)*

Problems:

- Cost of full search (at every step) can be prohibitive
- Search in adversarial environment
  - *Player will try to outsmart you*

© Alla Sheffer

Heuristic Search

- Blind search algorithms do not take into account the goal until they are at a goal node.
- Often there is extra knowledge that can be used to guide the search:
  - an estimate of the distance/cost from node *n* to a goal node.
- This estimate is called a search heuristic.

9

© Alla Sheffer

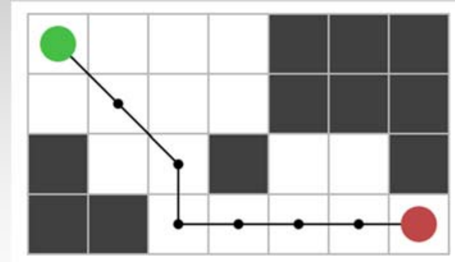
Best First Search (BestFS)

- Idea: always choose the path on the frontier with the smallest h value.
- BestFS treats the frontier as a priority queue ordered by h.
- Greedy approach: expand path whose last node seems closest to the goal - chose the solution that is locally the best.

© Alla Sheffer

Pathfinding

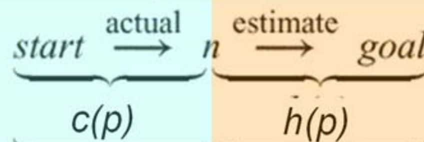
- How do I get from point A to point B?



© Alla Sheffer

A* Search

- A* search takes into account both
  - the **cost** of the path *p* to a node *c(p)*
  - the **heuristic value** of that path *h(p) (i.e. the h value of the node n at the end of p)*
- Let $f(p) = c(p) + h(p)$.
  - *f(p)* is an **estimate** of the cost of a path from the start to a goal via *p*.



A* always chooses the path on the frontier with the lowest **estimated** distance from the start to a goal node constrained to go via that path.

© Alla Sheffer

A* implementation

- 1. Initialize open, closed lists. Put starting node on open list.
- 2. While open list is not empty:
  - Find node with smallest f on the list, call it q
  - Pop q off of open list
  - Find q's "successors", and set their parent nodes to q

© Alla Sheffer

A* implementation

- 1. Initialize open, closed lists. Put starting node on open list.
- 2. While open list is not empty:
  - Find node with smallest f on the list, call it q
  - Pop q off of open list
  - Find q's "successors", and set their parent nodes to q

  - For each successor:
    - If successor is the goal, done!
    - g(successor) = g(q) + d(q,successor)
h(successor) = D(goal, successor)
    - If successor already exists in open list with lower f, skip it
    - If successor already exists in closed list with lower f, skip it
    - Otherwise, add successor to open list

© Alla Sheffer

Copyright: Alla Sheffer

A* implementation

- 1. Initialize open, closed lists. Put starting node on open list.
- 2. While open list is not empty:
  - Find node with smallest f on the list, call it q
  - Pop q off of open list
  - Find q's "successors", and set their parent nodes to q
  - For each successor:
    - If successor is the goal, done!
    - g(successor) = g(q) + d(q,successor)
      h(successor) = d(goal, successor)
    - If successor already exists in open list with lower f, skip it
    - If successor already exists in closed list with lower f, skip it
    - Otherwise, add successor to open list
  - Put q on closed list

© Alla Sheffer

A* search

Key idea: D is a heuristic, and not the real distance:

D(p,q) = |(p.x – q.x)| + |(p.y – q.y)|

- Manhattan distance

D(p,q) = sqrt((p.x – q.x)^2 + (p.y – q.y)^2)

- Euclidean distance

© Alla Sheffer

Copyright: Alla Sheffer

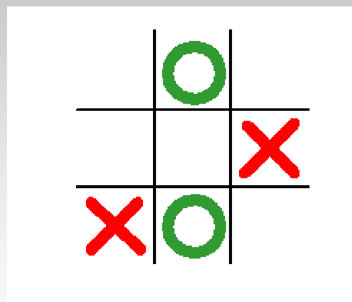
Min-Max Trees

- Adversarial planning in a turn-taking environment
  - *Algorithm seeks to maximize our success **F***
  - *Adversary seeks to minimize **F***
- Key idea: at each step algorithm selects move that minimizes highest (estimated) value of F adversary can reach
  - *Assume the opponent does what looks best*

© Alla Sheffer

Example

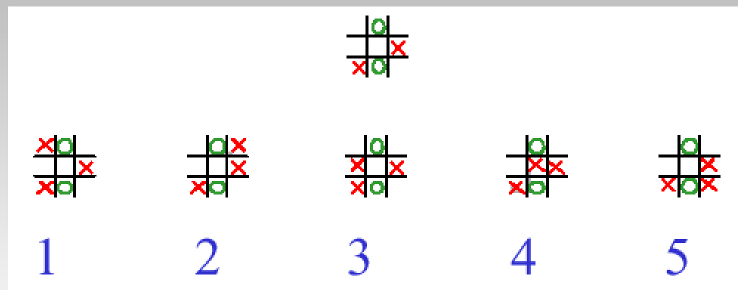
(from *uliana.lecturer.pens.ac.id/Kecerdasan%20Buatan/ppt/Game%20Playing/gametrees.ppt*)



We are playing X, and it is now our turn.

© Alla Sheffer

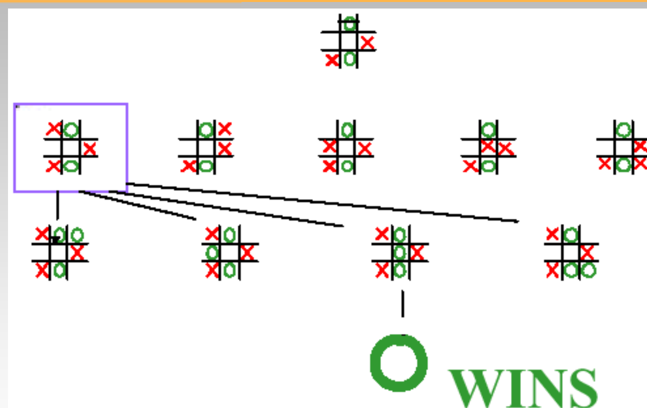
Our options:



Number = position after each legal move

© Alla Sheffer

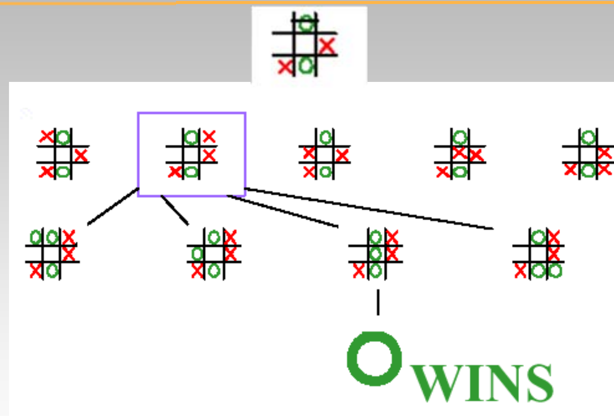
Opponent options



Here we are looking at all of the opponent responses to the first possible move we could make.

© Alla Sheffer

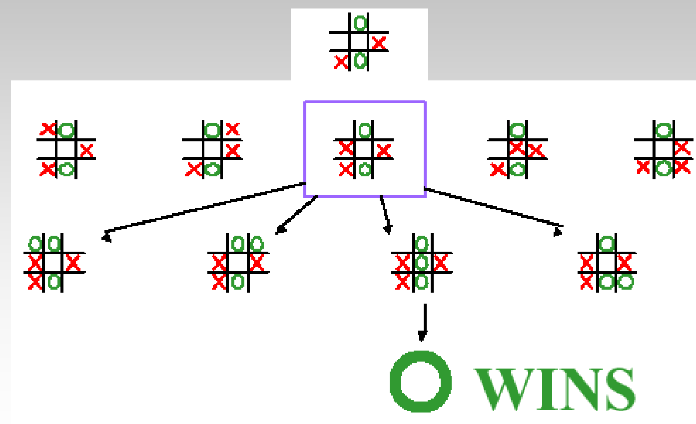
Opponent options



Opponent options after our second possibility. Not good again…

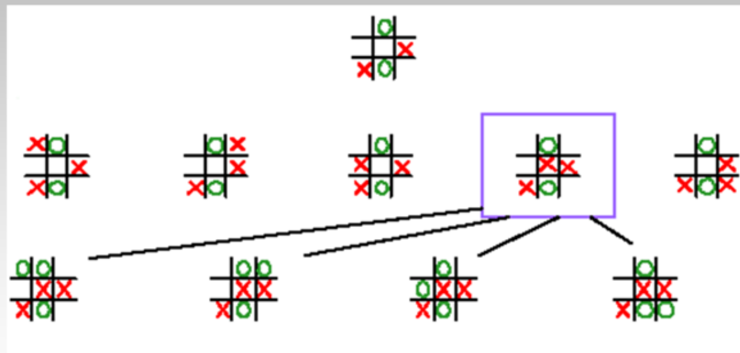
© Alla Sheffer

Opponent options



© Alla Sheffer

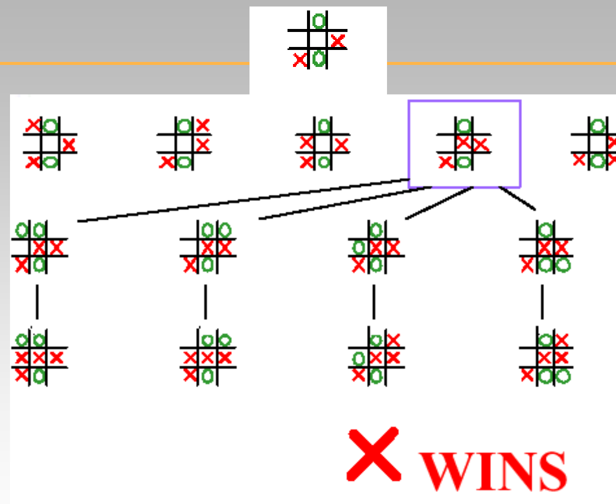
Opponent options => Our options



Now they don't have a way to win on their next move. So now we have to consider our responses to their responses.

© Alla Sheffer

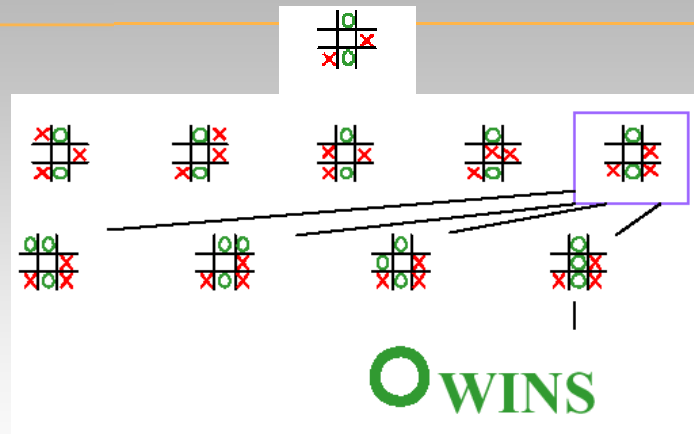
Our options



We have a win for any move they make. Original position in purple is an X win.

© Alla Sheffer

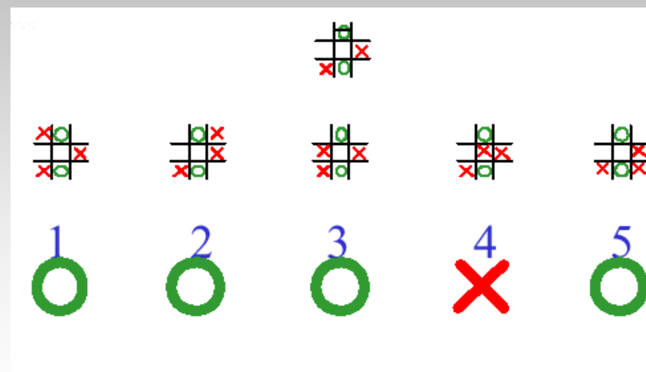
Other options



They win again if we take our fifth move.

© Alla Sheffer

Summary of the Analysis



So which move should we make? ;-)

© Alla Sheffer

MinMax algorithm

- Traverse "game tree":
  - *Enumerate all possible moves at each node.*
  - *The children of each node are the positions that result from making each move. A leaf is a position that is won or drawn for some side.*
- Assume that we pick the best move for us, and the opponent picks the best move for him (causes most damage to us)
- Pick the move that maximizes the minimum amount of success for our side.

© Alla Sheffer

MinMax Algorithm

- Tic-Tac-Toe: three forms of success: Win, Tie, Lose.
  - *If you have a move that leads to a Win make it.*
  - *If you have no such move, then make the move that gives the tie.*
  - *If not even this exists, then it doesn't matter what you do.*

© Alla Sheffer

Extensions

- Challenges: In practice
  - *Trees too deep/large to explore*
  - *Opponent no always makes the best choice*
  - *Randomness*
- Solution - Heuristics
  - *Rate nodes based on local information.*
  - *For example, in Chess "rate" a position by examining difference in number of pieces*

© Alla Sheffer

Heuristics in MinMax

- Strategy that will let us cut off the game tree at fixed depth (layer)
- Apply heuristic scoring to bottom layer
  - *instead of just Win, Loss, Tie, we have a score.*
- For "our" level of the tree we want the move that yields the node (position) with highest score. For a "them" level "they" want the child with the lowest score.

© Alla Sheffer

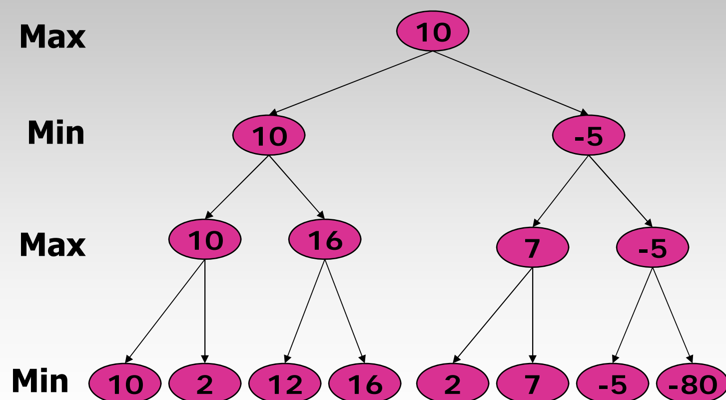
Pseudocode

```
int Minimax(Board b, boolean myTurn, int depth) {
      if (depth==0)
            return b.Evaluate(); // Heuristic
      for(each possible move i)
            value[i] = Minimax(b.move(i), !myTurn,
depth-1);
      if (myTurn)
            return array_max(value);
      else
            return array_min(value);
}
```

Note: we don't use an explicit tree structure.
However, the pattern of recursive calls forms a tree on the call stack.

© Alla Sheffer

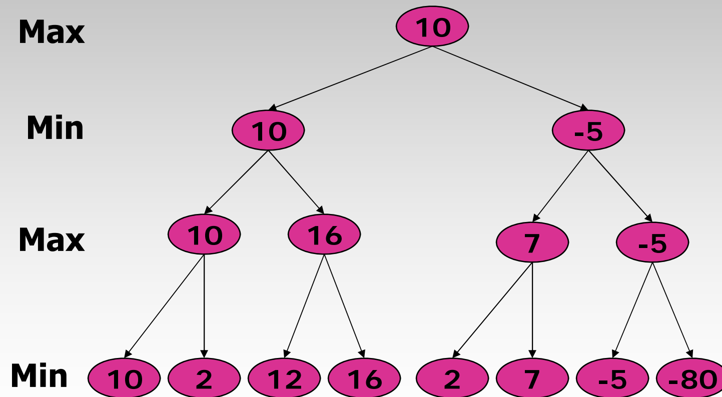
Real Minimax Example



Evaluation function applied to the leaves!

© Alla Sheffer

Pruning



© Alla Sheffer

Alpha Beta Pruning

Idea: Track "window" of expectations.

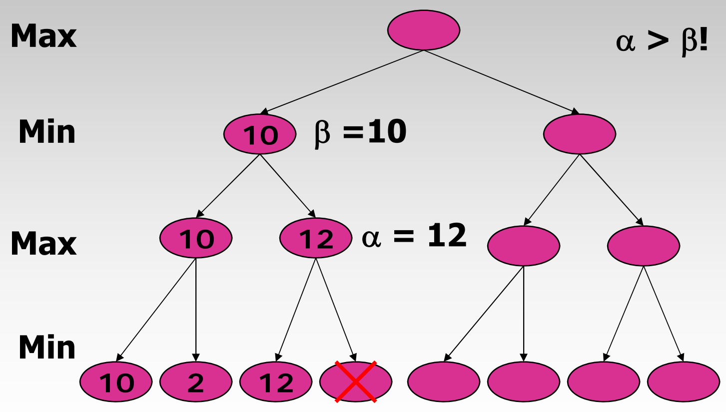
Use two variables

- α – Best score so far at a **max** node: increases
  - *At a child **min** node:*
    - Parent wants **max**. To affect the parent's current α, our β cannot drop below α.
  - *If β ever gets less:*
    - Stop searching further subtrees *of that child*. They do not matter!
- β – Best score so far at a **min** node: decreases
  - *At a child **max** node.*
    - Parent wants **min**. To affect the parent's current β, our α cannot get above the parent's β.
  - *If α gets bigger than β:*
    - Stop searching further subtrees *of that child*. They do not matter!

Start with an infinite window ($\alpha = -\infty$, $\beta = \infty$)

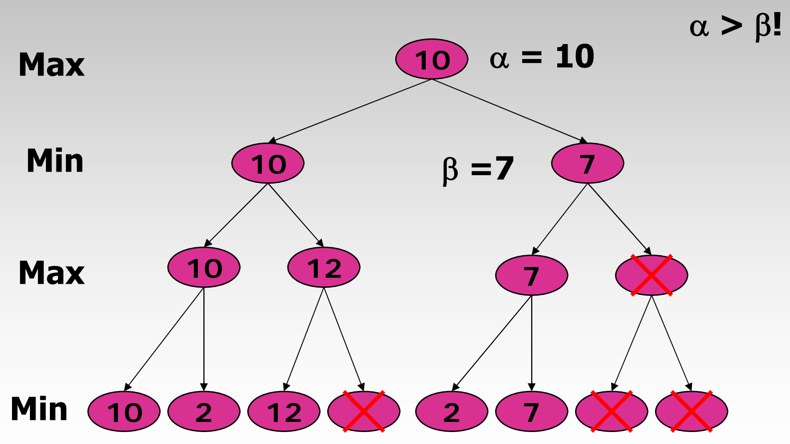
© Alla Sheffer

Alpha Beta Example



© Alla Sheffer

Alpha Beta Example



© Alla Sheffer

Pseudo Code

```
int AlphaBeta(Board b, boolean myTurn, int depth, int alpha, int beta) {
   if (depth==0)
      return b.Evaluate(); // Heuristic
   if (myTurn) {
      for(each possible move i && alpha < beta)
         alpha  = max(alpha,AlphaBeta(b.move(i),!myTurn,depth-1,alpha,beta));
      return alpha;
   }
   else {
      for(each possible move i && alpha < beta)
         beta  = min(beta,AlphaBeta(b.move(i), !myTurn, depth-1,alpha,beta));
      return beta;
   }
}
```

© Alla Sheffer