# AI

Week 3
Adversarial Search

# Games

- Most AI game playing is in games with the following characteristics
  - Zero-sum
  - Two player
  - Perfect Information
- Games that do not fit?

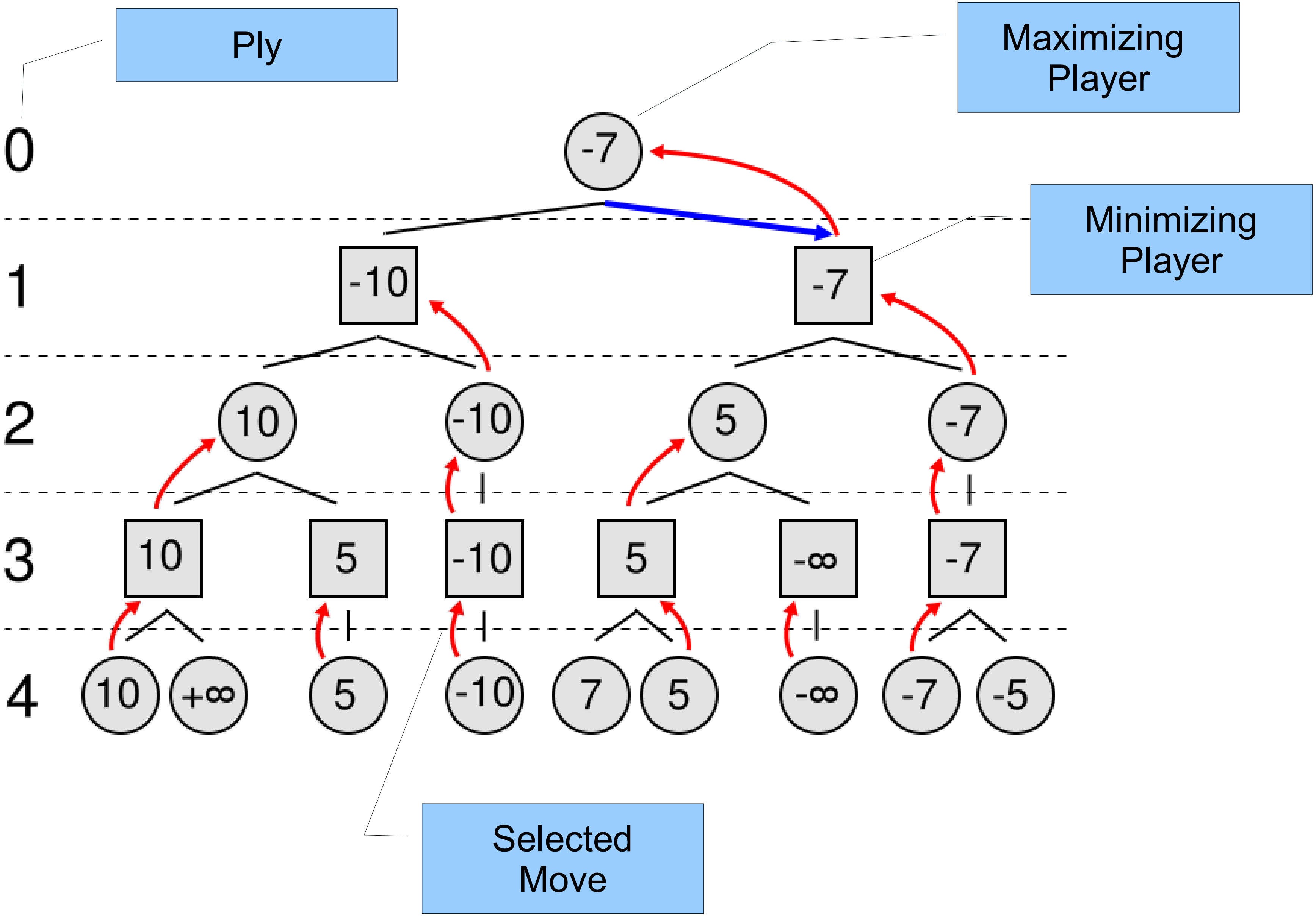
# Static Evaluation Function

- A heuristic – look at a board and estimate the probability of winning
- Useful when entire space cannot be searched
- Premise
  - The deeper you search, the better your estimate
  - Gaming terminology -- “ply”==depth==number of moves
  - So to get a better estimate
    - Search deeper
    - Get a better static evaluation function
    - learning?

# Search in a 2 player game

- Problem, the opponent always gets to move
- So, every other move in the search tree is made by the opposition
- Solution: “minimax algorithm”
  - Assume: both players play optimally
  - Assume: opponent evaluates the board exactly as you
  - Under these two assumptions opponent will always choose what is best for them, which is the worst for you

# Minimax Algorithm



# Minimax Algorithm

```
int MinMax(int depth) {
    if (SideToMove() == WHITE)     return Max(depth);
    else    return Min(depth);
}
```

```
int Max(int depth)   {
    int best = -INFINITY;
    if (depth <= 0)  return Evaluate();
    GenerateLegalMoves();
    while (MovesLeft()) {
        MakeNextMove();
        val = Min(depth - 1);
        UnmakeMove();
        if (val > best)          best = val;
    }
    return best;
}
```

```
int Min(int depth){
    int best = INFINITY;
    if (depth <= 0)   return Evaluate();
    GenerateLegalMoves();
    while (MovesLeft()) {
        MakeNextMove();
        val = Max(depth - 1);
        UnmakeMove();
        if (val < best)          best = val;
    }
    return best;
}
```

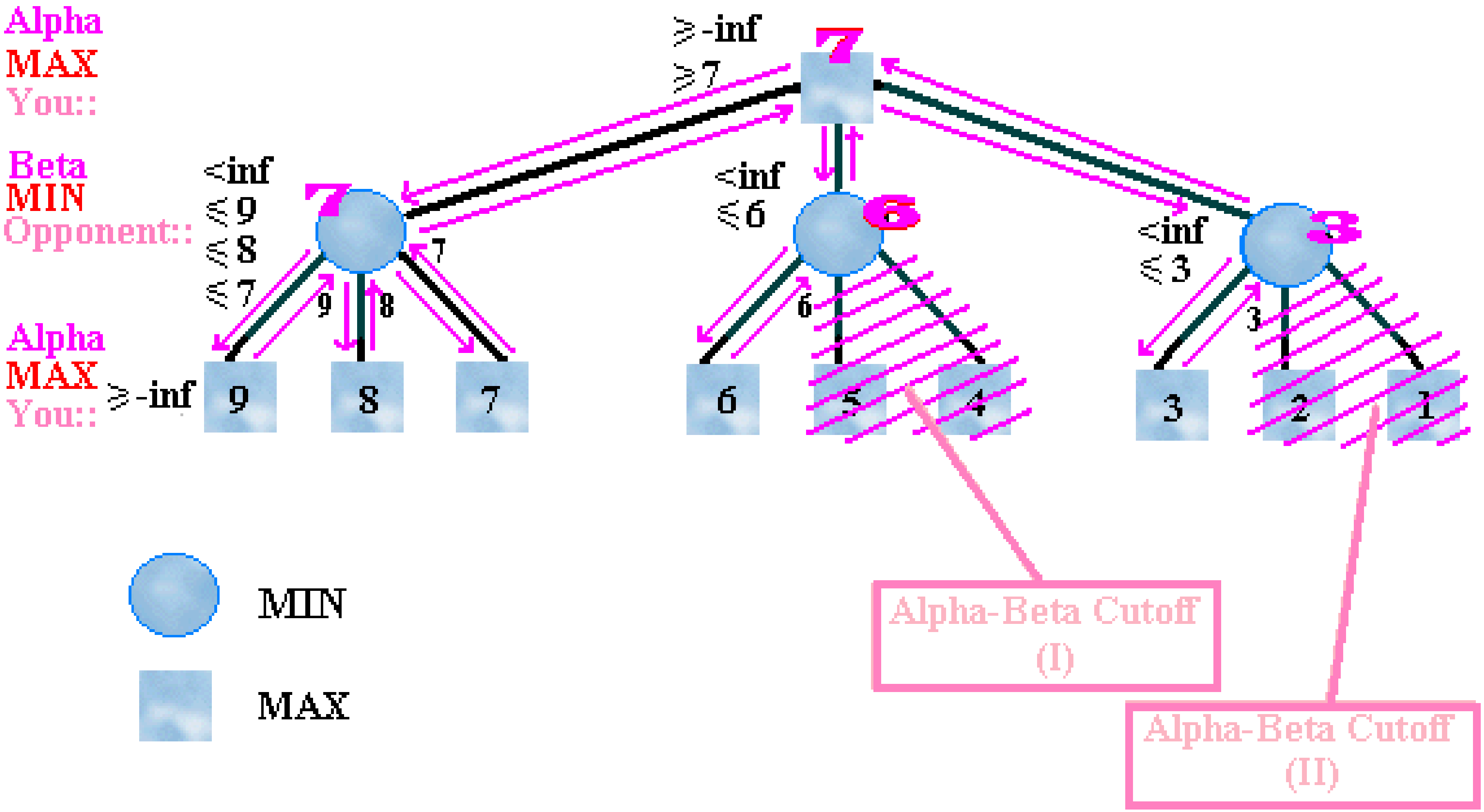
# More on minimax: Problems

- 3 (or more) players?
- Games that are not zero sum?
- Branching factor in chess mid games is about 35 so how deep can you afford to search?
  - Do you have to look at all 35 branches?
    - Experts do not
    - Some chess specific tricks get you down to about 3-5.
    - Can you eliminate branches without knowing chess?

# Alpha-Beta Pruning

- Basic Idea, do not expand any nodes that you know would never be used
- While doing minimax search keep two numbers
  - Alpha – the best score that you can get
  - Beta – the worst move that the opponent will allow

# Alpha-Beta Pruning Example



# Alpha-Beta Pruning  Algorithm

```
int AlphaBeta(int depth, int alpha, int beta)  {
    if (depth == 0)         return Evaluate();
    GenerateLegalMoves();
    while (MovesLeft()) {
        MakeNextMove();
        val = -AlphaBeta(depth - 1, -beta, -alpha);
        UnmakeMove();
        if (val >= beta)            return beta;
        if (val > alpha)            alpha = val;
    }
    return alpha;
}
```

# Alpha-Beta Pruning Conclusions

- Best case
  - Need to examine only square root of number of nodes
  - This would give you the time to search twice as deep
- Problem
  - To get best case need to carefully pick the order of nodes to be expanded
- Average case
  - About half of theoretical max
- Horizon effect

# Backgammon

- Problems
  - Does not fit “2 player, perfect info, zero sum”
  - Dice give non-determinism and have effect of raisin branching factor
    - Mid game branching factor easily exceeds 100
  - So, what to do?
- Traditional Answer
  - Hand craft static evaluation function
  - Search like mad

# Neurogammon

- Idea:
  - Do not hand craft a static evaluation function – learn it using a neural network.
    - Neural networks use math to address credit assignment problem
  - 1 move lookahead "if I do X, how good is it"
  - Train NN using a library of 300,000 board positions
    - Take two alternate moves from a given board and expert says which is better
- Created a "strong intermediate" player

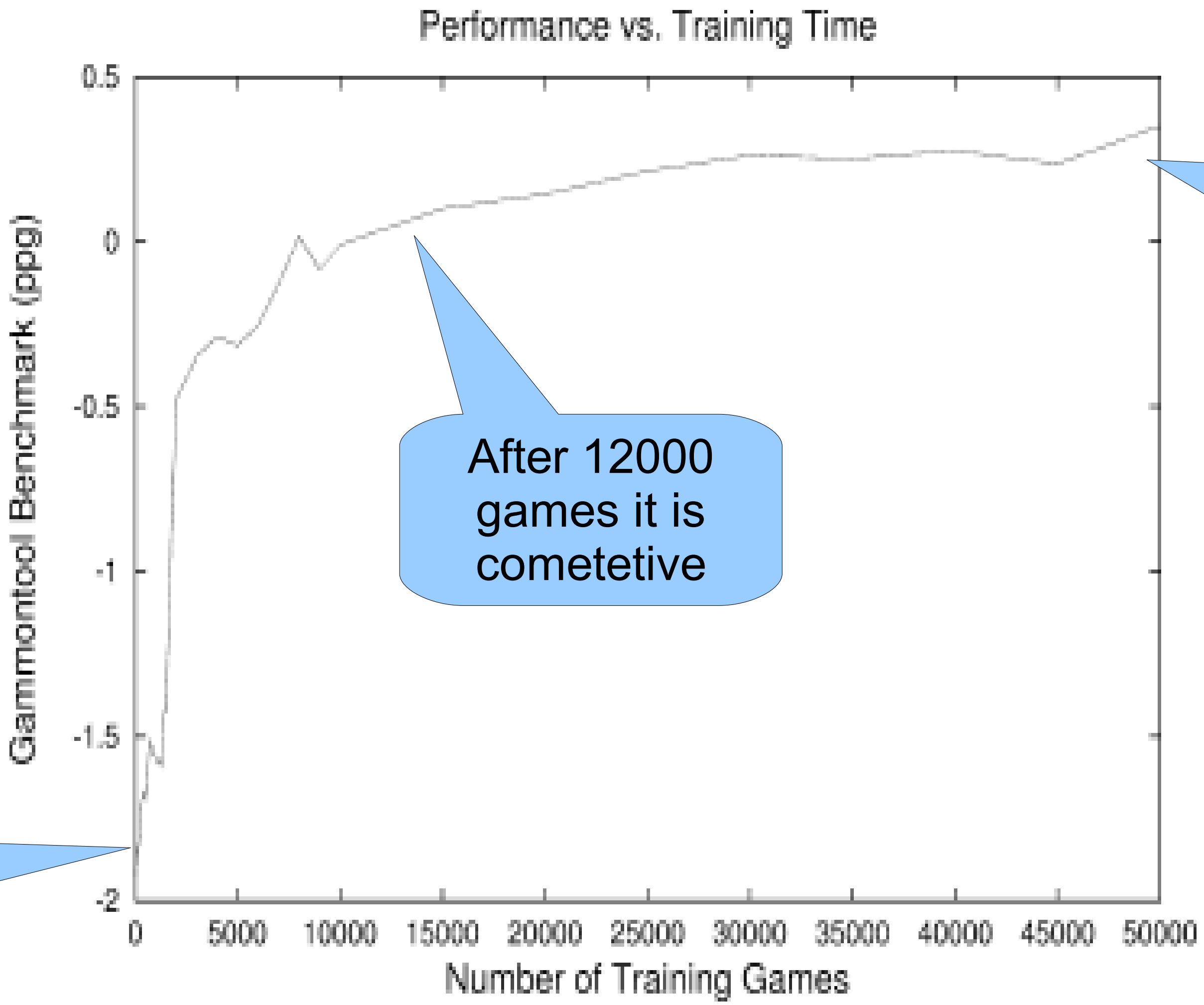
# NeuroGammon -- analysis

- Made a lot of poor moves
  - Insufficient training dataset?
    - Problem: experts get bored
- Essentially learned to replicate the play of the expert who rated the moved
  - Without deep search computers can usually get slightly better than their programmers at games, but only slightly
    - This was true of neurogammon
    - Why?

# TD Gammon

- Idea:
  - Have computer play against itself
  - No programmed knowledge other than rules of the game
  - Initially makes almost random moves
  - But it gets better!
- Problem
  - The credit assignment problem still
  - Also, which move in a sequence deserves credit
- It seems odd that is works
  - Start with a really bad player playing against itself and over time it becomes an expert!

# TD Gammon - 1.0



Initially gets gammoned a lot

After 12000 games it is cometetive

Sill improving

Performance vs a good traditional backgammon program
About competetive with neurogammon

# TD Gammon 2 - 3

- Deeper look ahead
- More training games
- Achieved expert level play
- Of note
  - Experts play is now heavily influenced by computer
  - Experts regularly practice against computers
  - TD gammon was first AI to win a world championship at any game

# AI Terms

- Zero Sum Game
- Perfect Information
- Static Evaluation Function
- Ply
- Minimax algorithm
- Alpha Beta Pruning
- Horizon Effect