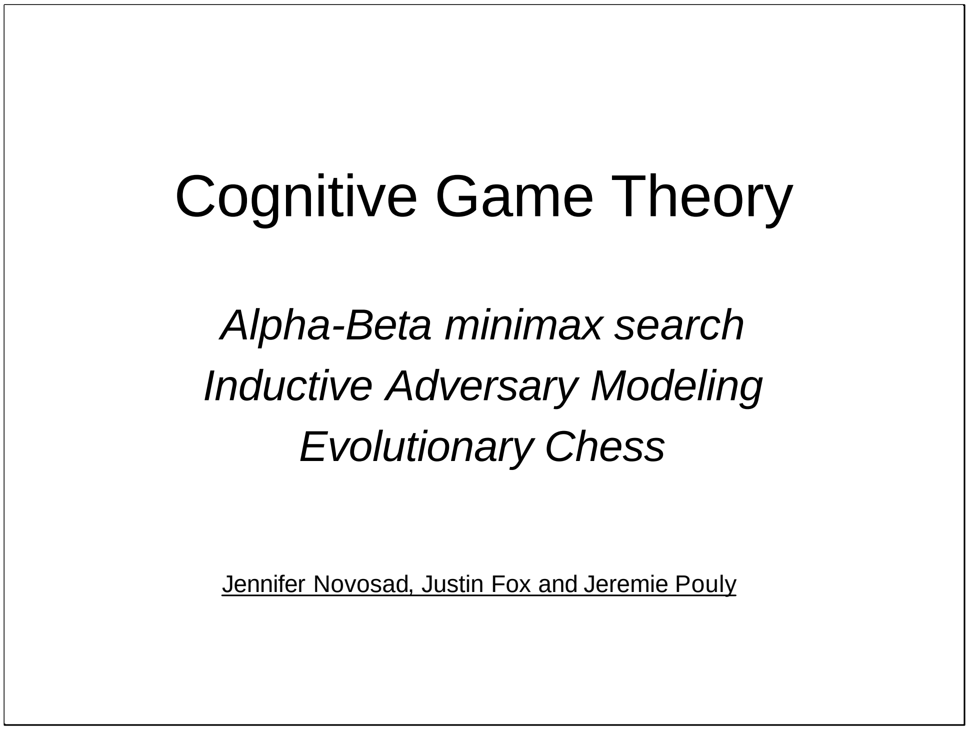


Cognitive Game Theory

Alpha-Beta minimax search
Inductive Adversary Modeling
Evolutionary Chess

Jennifer Novosad, Justin Fox and Jeremie Pouly

Our lecture topic is cognitive game.

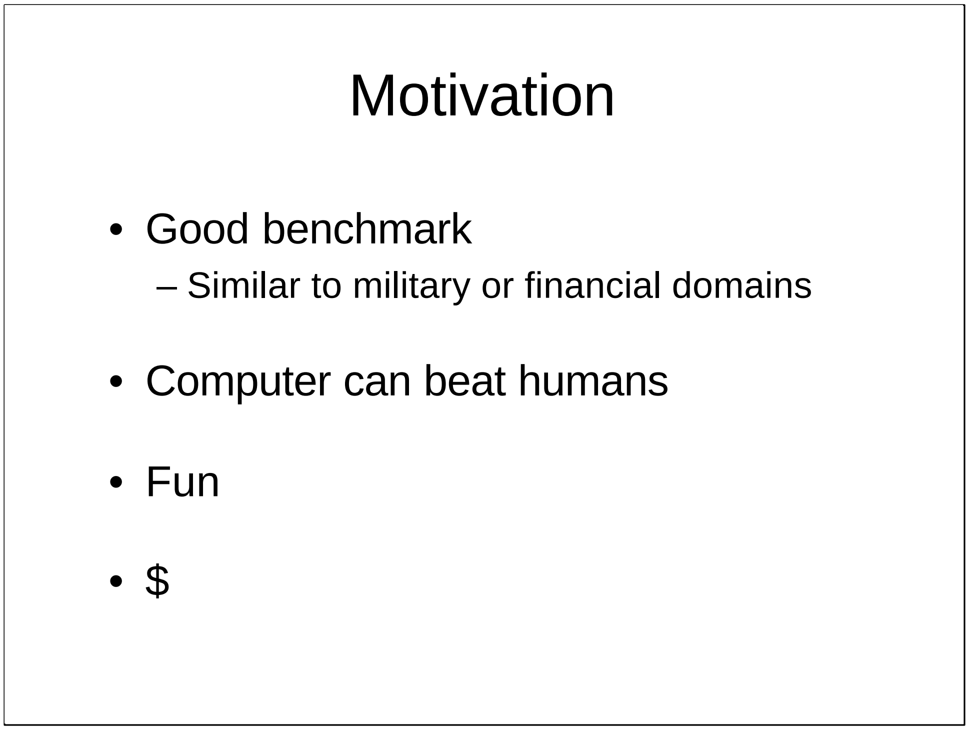
We are interested in this subject because games are a simple representation of reality on which we can test any concept developed in artificial intelligence. For this reason games have always been considered as an attractive framework for new developments.

Our talk in divided in three parts:

• Jeremie will first give a quick review of the minimax search and present a few improvements including alpha-beta cutoffs, transposition table and move ordering. He will also introduce the two demonstrations of the lecture.

• Jennifer

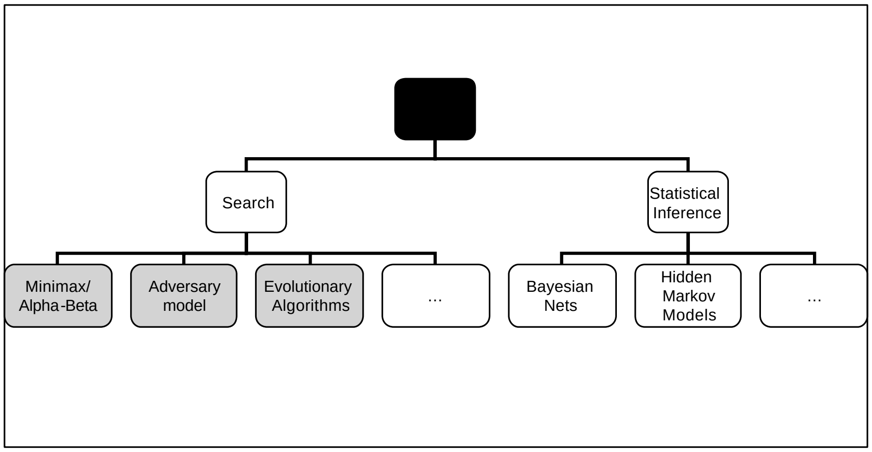
• Justin



Motivation

- Good benchmark
  - Similar to military or financial domains
- Computer can beat humans
- Fun
- $

Reasoning Techniques for Games



Search

Statistical Inference

Minimax/ Alpha-Beta

Adversary model

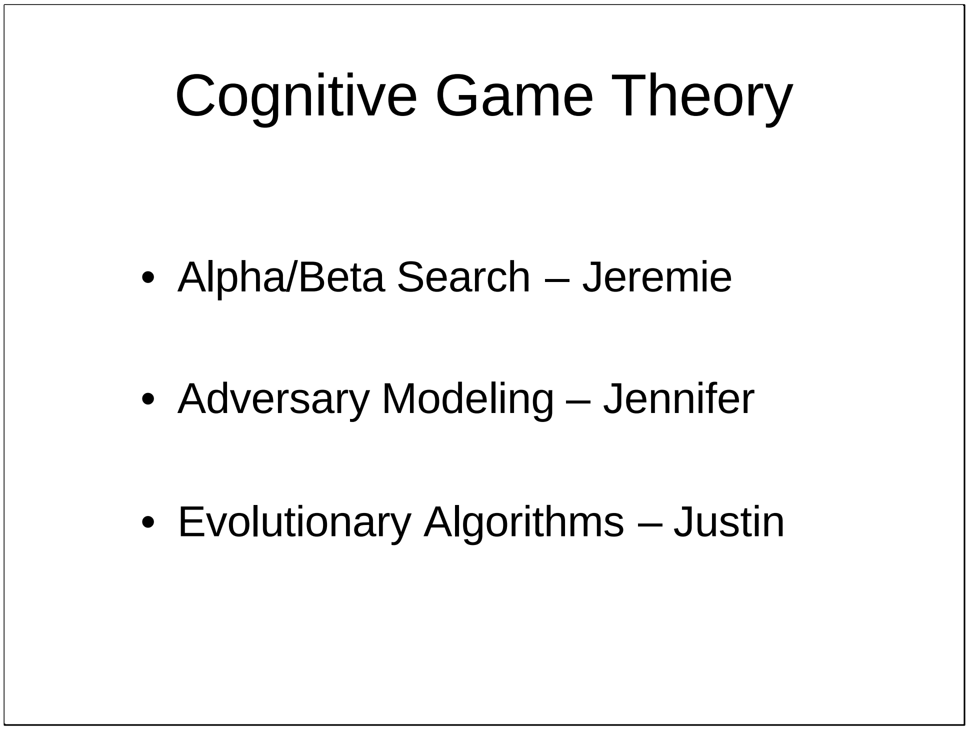
Evolutionary Algorithms

…

Bayesian Nets

Hidden Markov Models

…

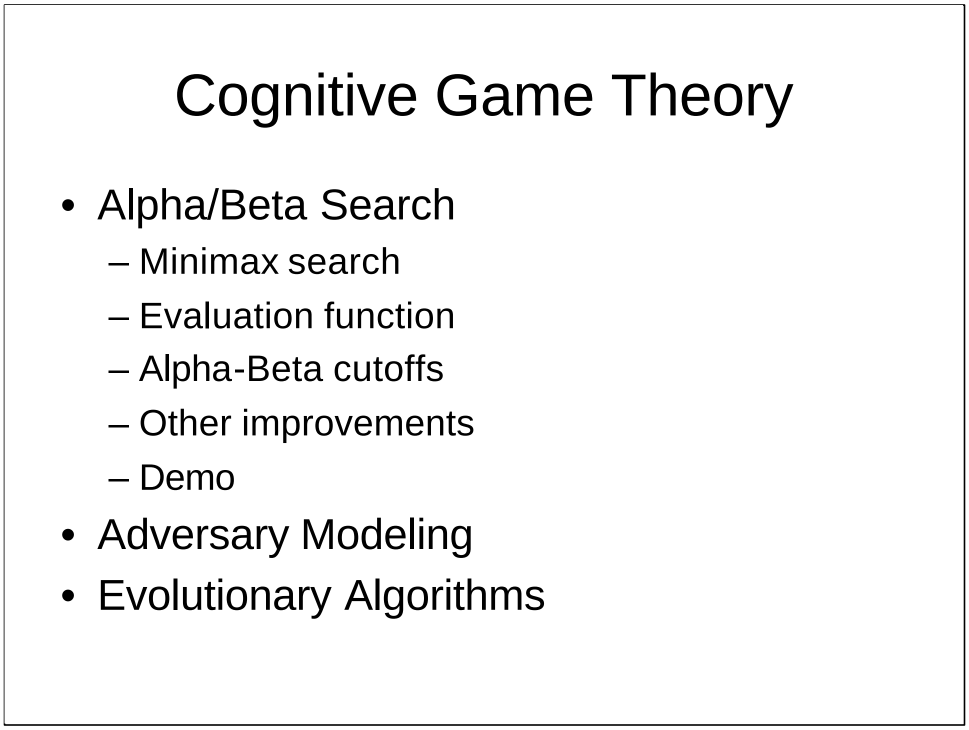


Cognitive Game Theory

- Alpha/Beta Search – Jeremie
- Adversary Modeling – Jennifer
- Evolutionary Algorithms – Justin

We return to the outline to note that the next section of this talk will now focus on a still small, but more detailed and less abstract example of how evolutionary algorithms may be applied to create chess players. This example can be found in the paper:

Kendall and Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics*, Proc. 2001 IEEE Congress on Evolutionary Computation.



Cognitive Game Theory

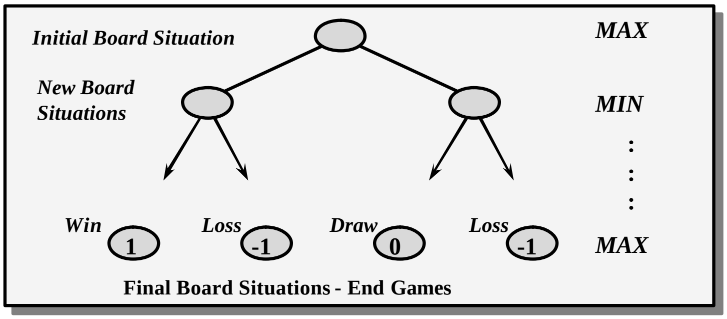
- Alpha/Beta Search
  - Minimax search
  - Evaluation function
  - Alpha-Beta cutoffs
  - Other improvements
  - Demo
- Adversary Modeling
- Evolutionary Algorithms

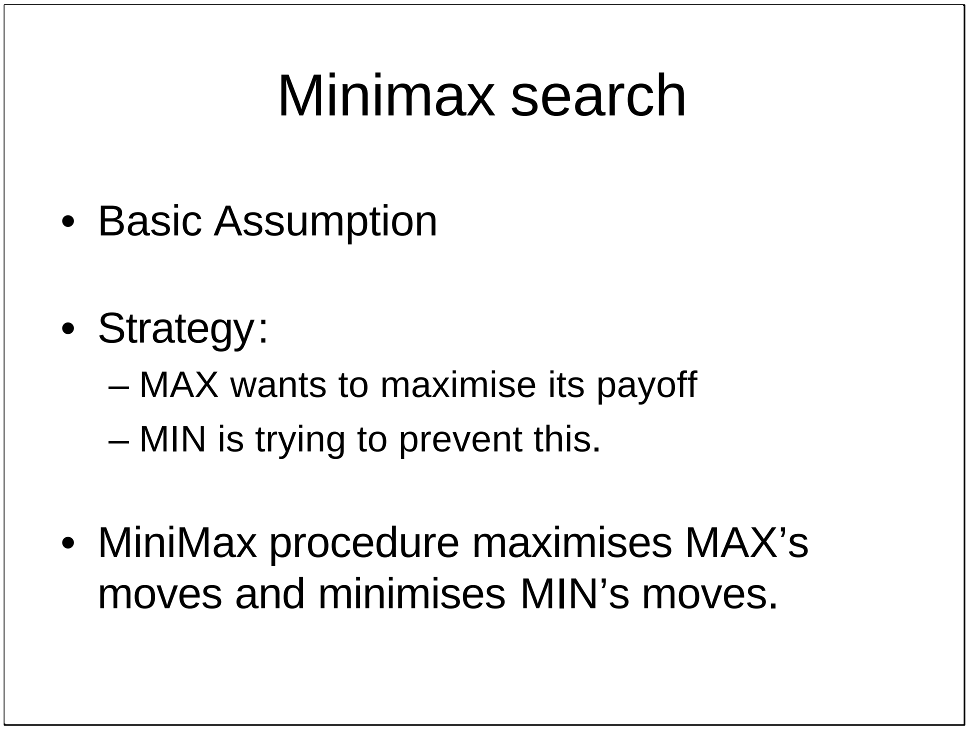
We return to the outline to note that the next section of this talk will now focus on a still small, but more detailed and less abstract example of how evolutionary algorithms may be applied to create chess players. This example can be found in the paper:

Kendall and Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics*, Proc. 2001 IEEE Congress on Evolutionary Computation.

Adversarial search

- Two-person games: Players = *Max* & *Min*
  - – Max wants to win
  - – Min wants Max to loose





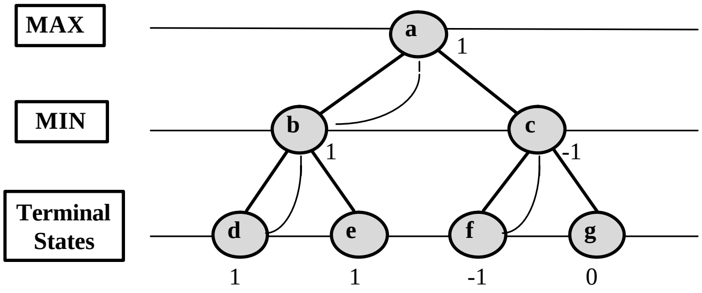
Minimax search

- Basic Assumption

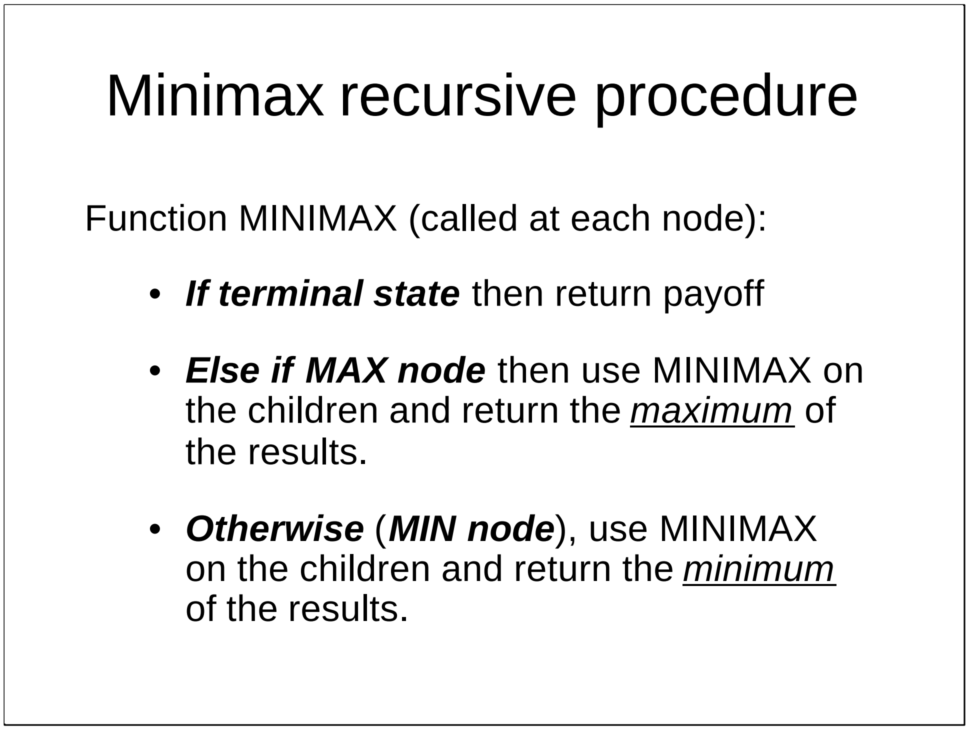
- Strategy:
  - MAX wants to maximise its payoff
  - MIN is trying to prevent this.

- MiniMax procedure maximises MAX’s moves and minimises MIN’s moves.

An example



Best value for MAX is 1



Minimax recursive procedure

Function MINIMAX (called at each node):

- ***If terminal state*** then return payoff
- ***Else if MAX node*** then use MINIMAX on the children and return the *maximum* of the results.
- ***Otherwise*** (***MIN node***), use MINIMAX on the children and return the *minimum* of the results.

Problems

- Time complexity: O(b^m)

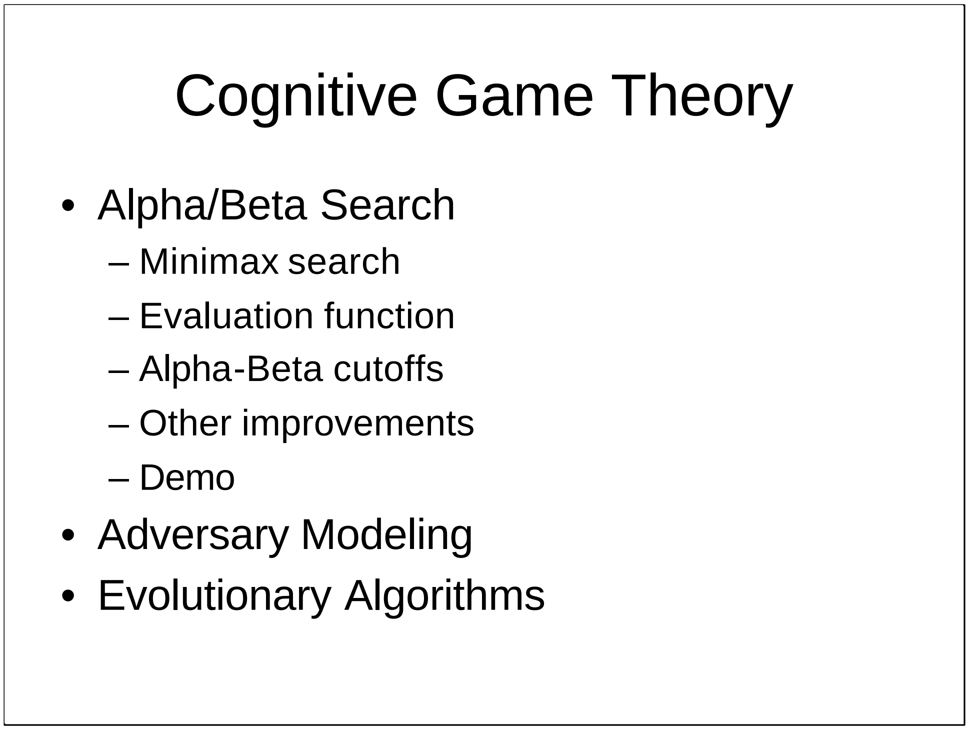
  b branching factor and m depth of the terminal states

  (Chess, b=35, m=100 → 35^{100}»10^{154} nodes to visit)

- Not possible to search the full game tree

  ⇨ Cutoff the tree at a certain depth

- But payoffs defined only at terminal states



Cognitive Game Theory

- Alpha/Beta Search
  - – Minimax search
  - – Evaluation function
  - – Alpha-Beta cutoffs
  - – Other improvements
  - – Demo
- Adversary Modeling
- Evolutionary Algorithms

We return to the outline to note that the next section of this talk will now focus on a still small, but more detailed and less abstract example of how evolutionary algorithms may be applied to create chess players. This example can be found in the paper:

Kendall and Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics*, Proc. 2001 IEEE Congress on Evolutionary Computation.

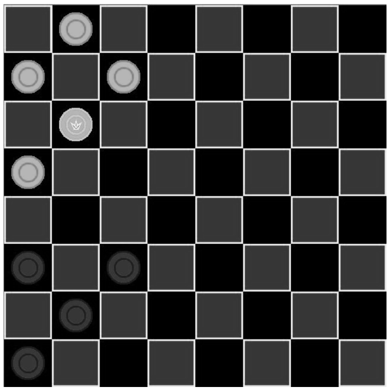
Heuristic evaluation function

- Estimate the chance of winning from board configuration.

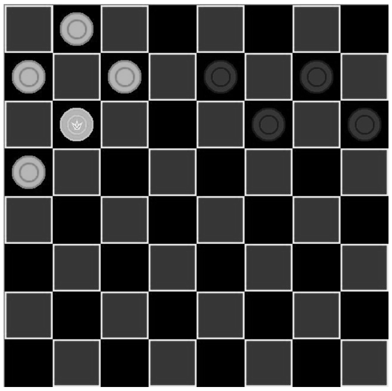
- Important qualities:
  - Must agree with terminal states
  - Must be fast to compute
  - Should be accurate enough

- Chess or checkers: Value of all white pieces – Value of all black pieces

Heuristic evaluation function



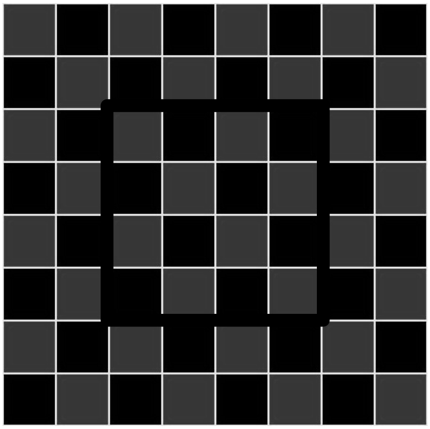
Val = (4*1) – (4*1+1*2) = -2



Val ???

Our evaluation function

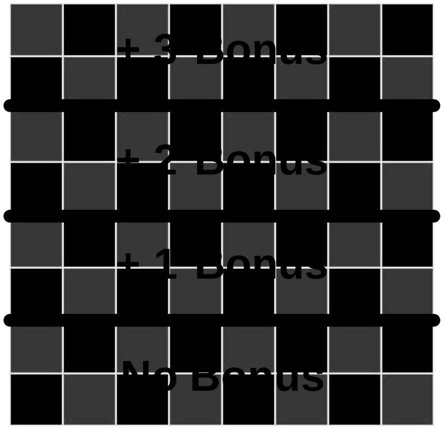
- Normal checker = 100000
- 4 parameters (long):
  - King value
  - ● Bonus central square for kings
  - Bonus move forward for checkers
  - Bonus for order of the moves (*depth/2)

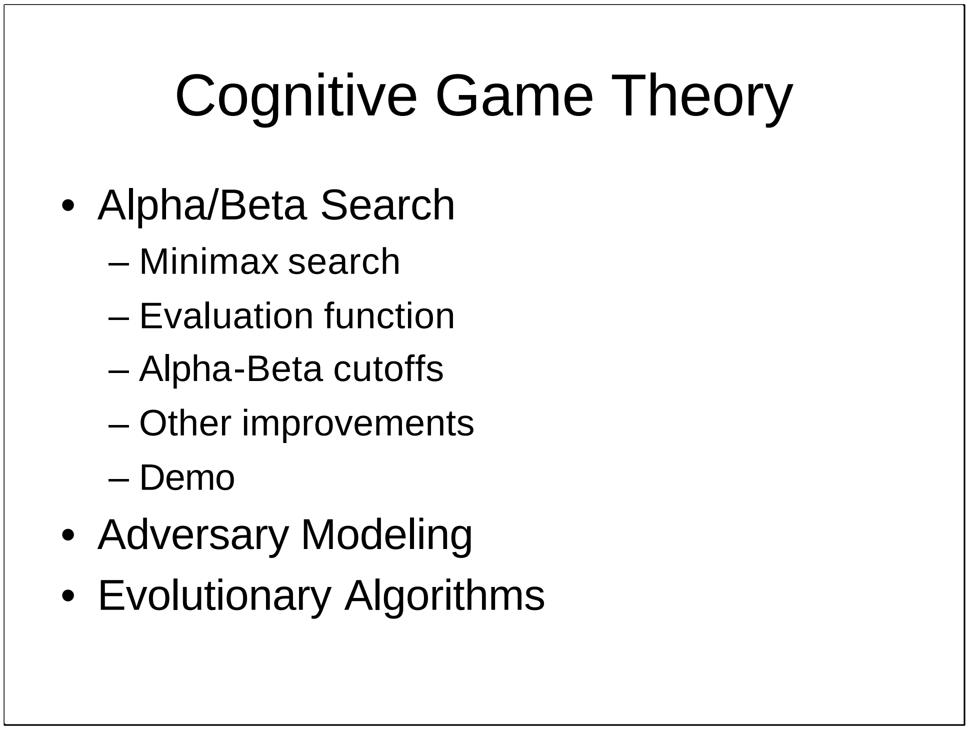


Our evaluation function

- Normal checker = 100000

- 4 parameters (long):
  – King value
  – Bonus central square for kings
  ⬬ Bonus move forward for checkers
  – Bonus for order of the moves (*depth/2)



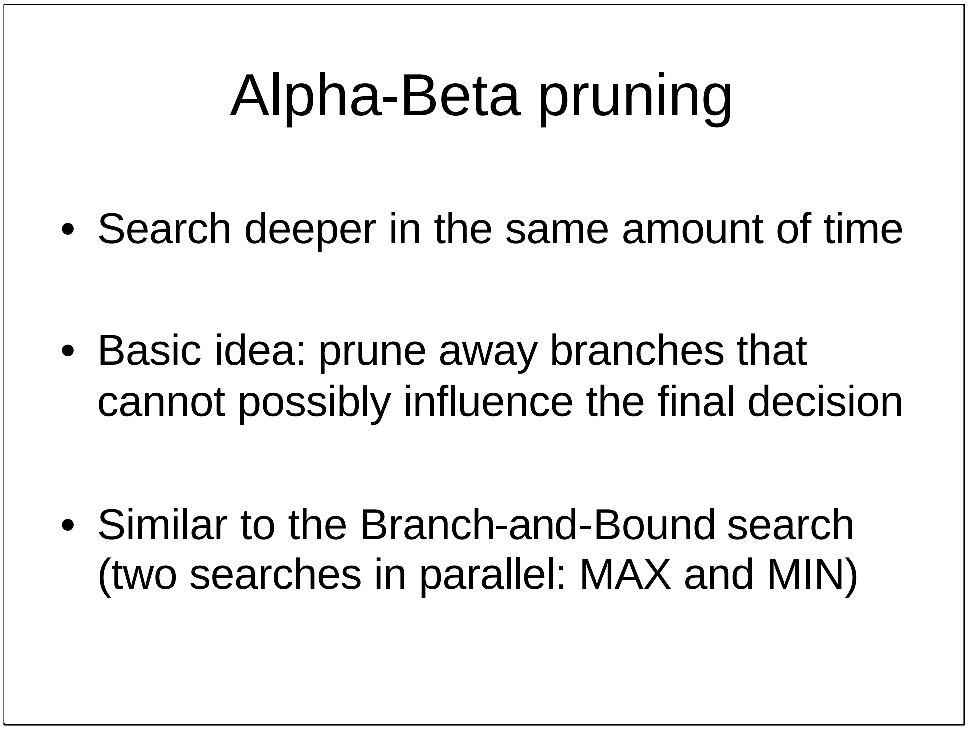


Cognitive Game Theory

- Alpha/Beta Search
  - – Minimax search
  - – Evaluation function
  - – Alpha-Beta cutoffs
  - – Other improvements
  - – Demo
- Adversary Modeling
- Evolutionary Algorithms

We return to the outline to note that the next section of this talk will now focus on a still small, but more detailed and less abstract example of how evolutionary algorithms may be applied to create chess players. This example can be found in the paper:

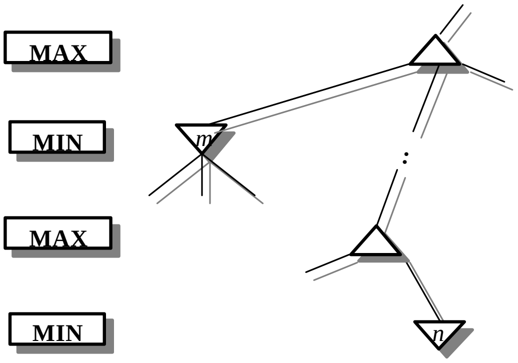
Kendall and Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics*, Proc. 2001 IEEE Congress on Evolutionary Computation.



Alpha-Beta pruning

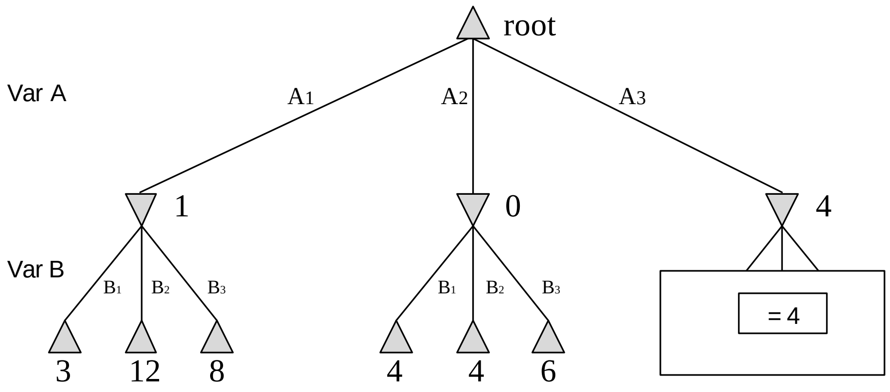
- Search deeper in the same amount of time
- Basic idea: prune away branches that cannot possibly influence the final decision
- Similar to the Branch-and-Bound search (two searches in parallel: MAX and MIN)

General case



If m is better than n for MAX then n will never get into play because m will always be chosen in preference.

Review of Branch-and-Bound



Best assignment: [A1,B1], value = 3

Alpha-Beta procedure

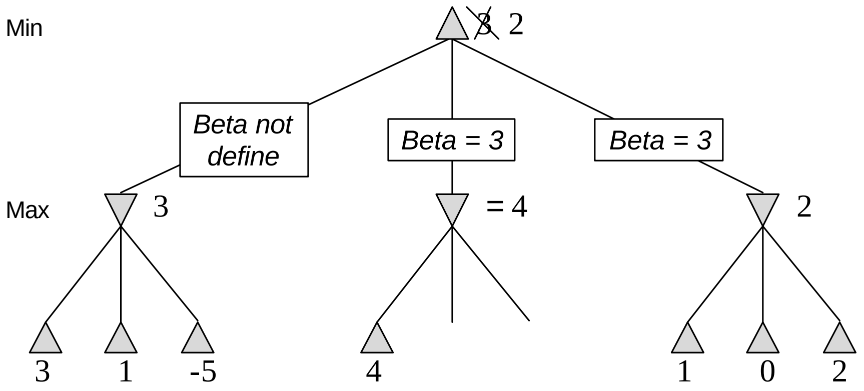
- Search game tree keeping track of:
  - Alpha: Highest value seen so far on maximizing level
  - Beta: Lowest value seen so far on minimizing level

- Pruning:
  - MAX node: prune parent if node evaluation smaller than Alpha
  - MIN node: prune parent if node evaluation greater than Beta

Branch-and-Bound analogy

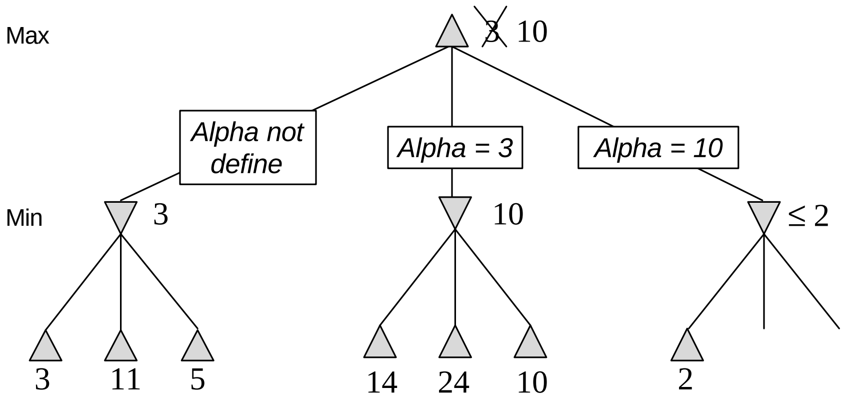
- MIN: minimize board valuation → minimize constraints in Branch-and-Bound
- MAX: maximize board valuation → inverse of Branch-and-Bound (but same idea)
- **Prune parent instead of current node** *(stop expanding siblings)*

Example MIN

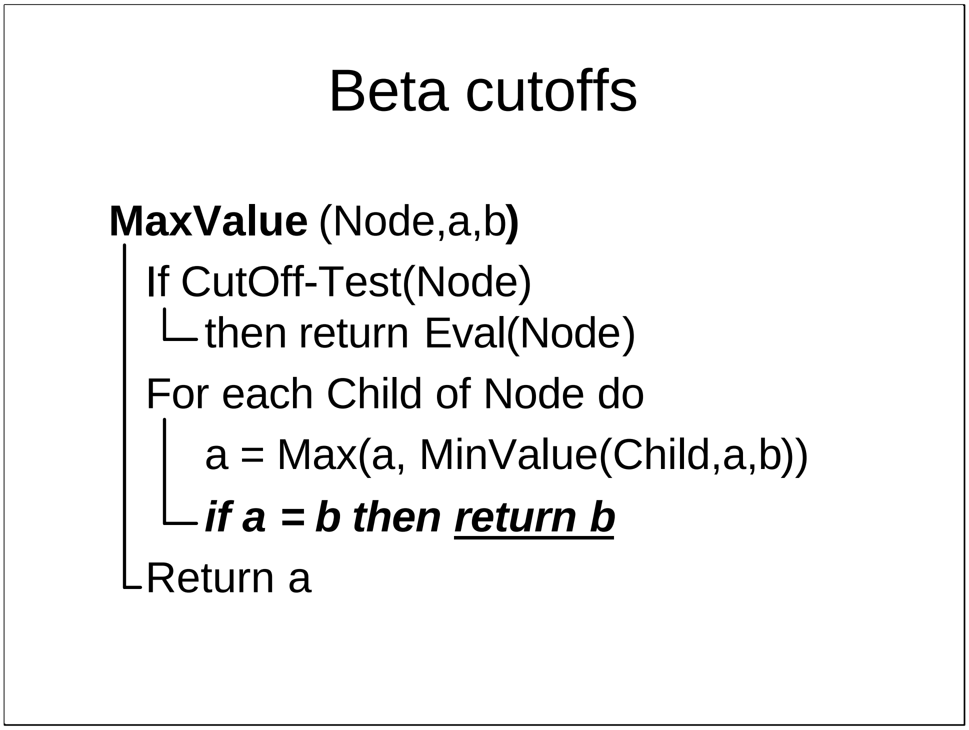


Beta: Lowest value seen so far on minimizing level

Example MAX

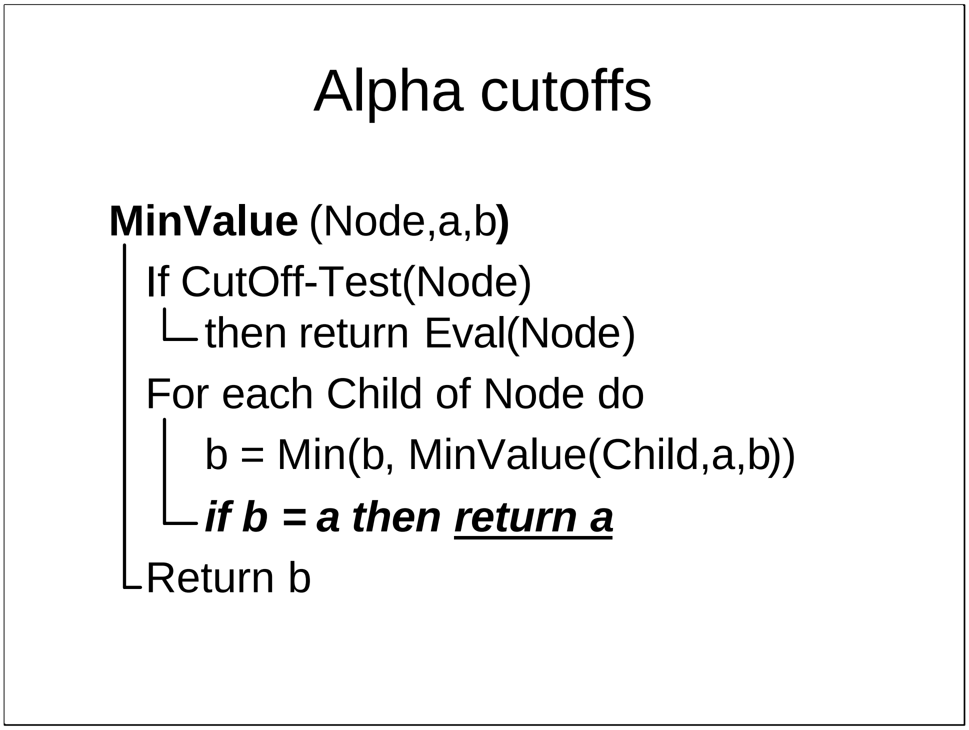


Alpha: Highest value seen so far on maximizing level



Beta cutoffs

```
MaxValue (Node,a,b)
    If CutOff-Test(Node)
        then return Eval(Node)
    For each Child of Node do
        a = Max(a, MinValue(Child,a,b))
        if a = b then return b
    Return a
```

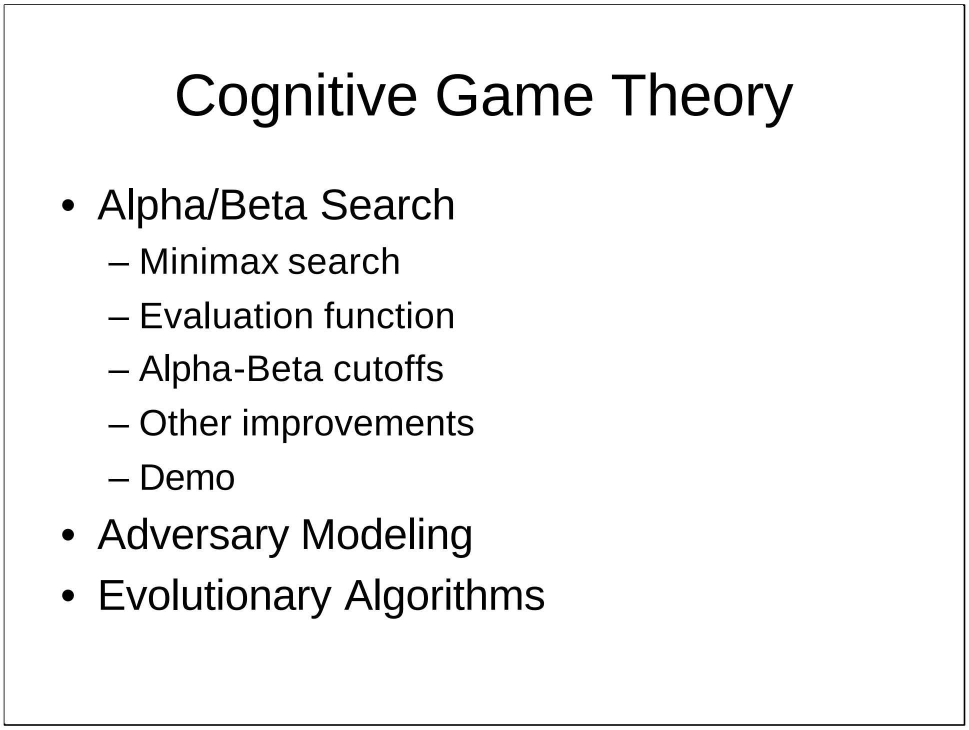


Alpha cutoffs

```
MinValue (Node,a,b)
    If CutOff-Test(Node)
        then return Eval(Node)
    For each Child of Node do
        b = Min(b, MinValue(Child,a,b))
        if b = a then return a
    Return b
```

Alpha-Beta gains

- Effectiveness depends on nodes ordering
- Worse case: no gain (no pruning) → O(b^d)
- Best case (best first search) → O($b^{d/2}$) i.e. allows to double the depth of the search!
- Expected complexity: O($b^{3d/4}$)

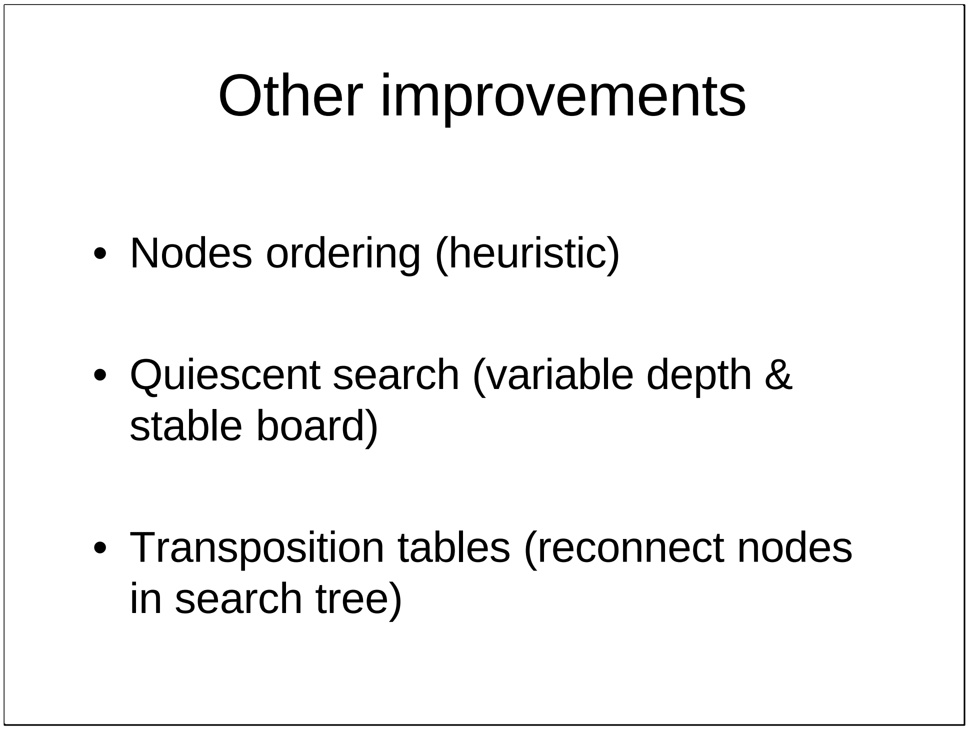


Cognitive Game Theory

- Alpha/Beta Search
  - – Minimax search
  - – Evaluation function
  - – Alpha-Beta cutoffs
  - – Other improvements
  - – Demo
- Adversary Modeling
- Evolutionary Algorithms

We return to the outline to note that the next section of this talk will now focus on a still small, but more detailed and less abstract example of how evolutionary algorithms may be applied to create chess players. This example can be found in the paper:

Kendall and Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics*, Proc. 2001 IEEE Congress on Evolutionary Computation.



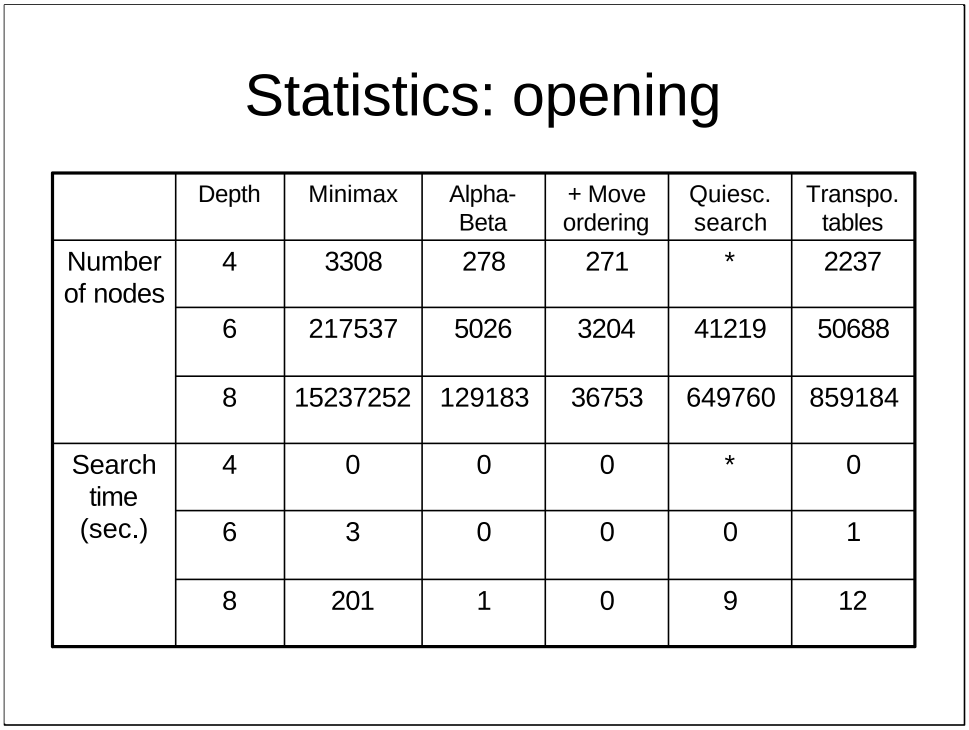
Other improvements

- Nodes ordering (heuristic)
- Quiescent search (variable depth & stable board)
- Transposition tables (reconnect nodes in search tree)

Advanced algorithm

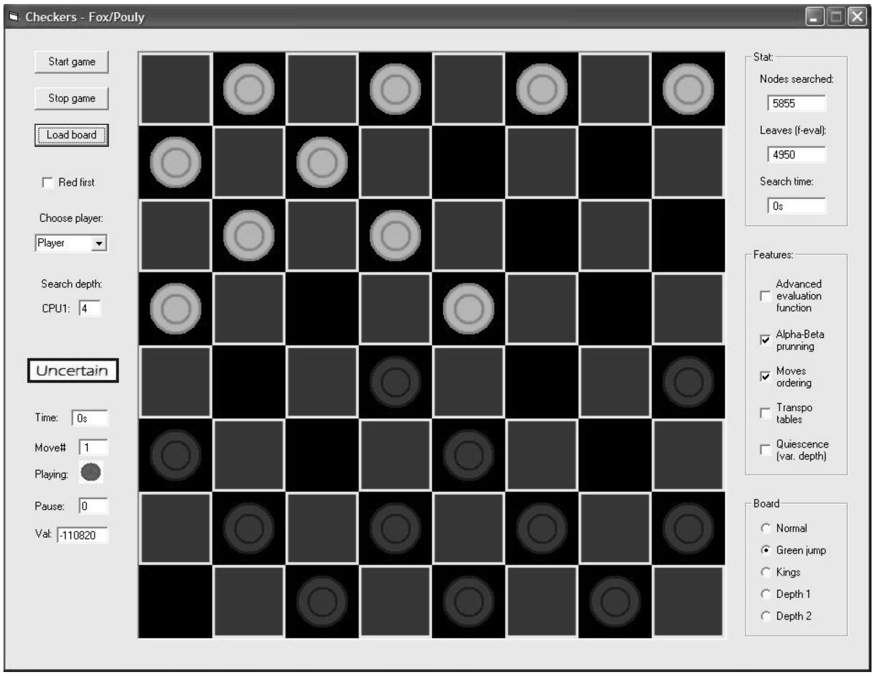
MaxValue (Node,a,b**)**

```
If board already exist in transposition tables then
    if new path is longer return value in the table
Save board in transposition table
If CutOff-Test(Node) then
    if quiescent board then return Eval(Node)
Find all the children and order them (best first)
For each Child of Node (in order) do
    a:=Max(a,MinValue(Child,a,b))
    if a>=b then return b
Return a
```



Statistics: opening

	Depth	Minimax	Alpha-Beta	+ Move ordering	Quiesc. search	Transpo. tables
Number of nodes	4	3308	278	271	*	2237
	6	217537	5026	3204	41219	50688
	8	15237252	129183	36753	649760	859184
Search time (sec.)	4	0	0	0	*	0
	6	3	0	0	0	1
	8	201	1	0	9	12



□ Red first

Choose player: Player

Search depth:
CPU1: 4

Time: 0s

Move# 1

Playing: ●

Pause: 0

Val: -110820

Stat:

Nodes searched: 5855

Leaves (f-eval): 4950

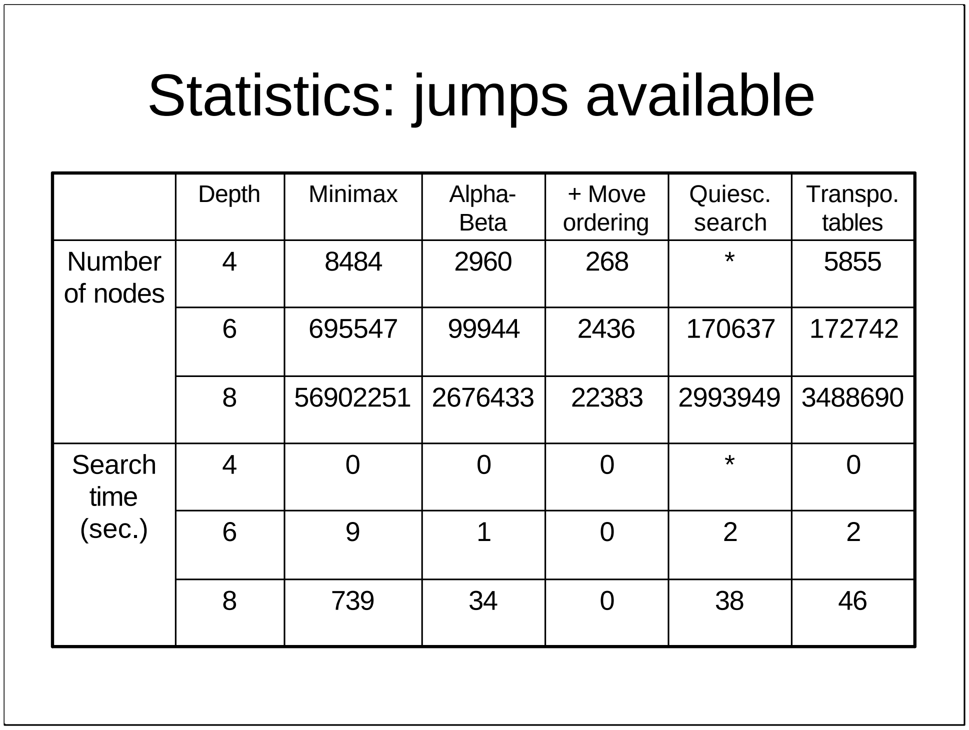
Search time: 0s

Features:

- □ Advanced evaluation function
- ☑ Alpha-Beta prunning
- ☑ Moves ordering
- □ Transpo tables
- □ Quiescence (var. depth)

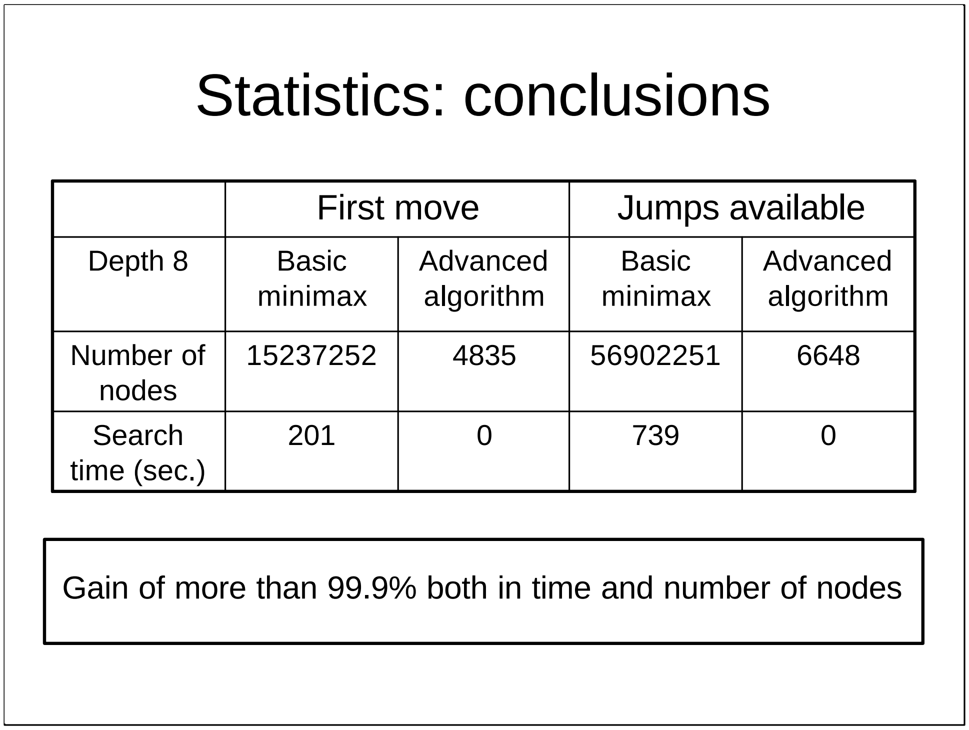
Board

- ○ Normal
- ● Green jump
- ○ Kings
- ○ Depth 1
- ○ Depth 2



Statistics: jumps available

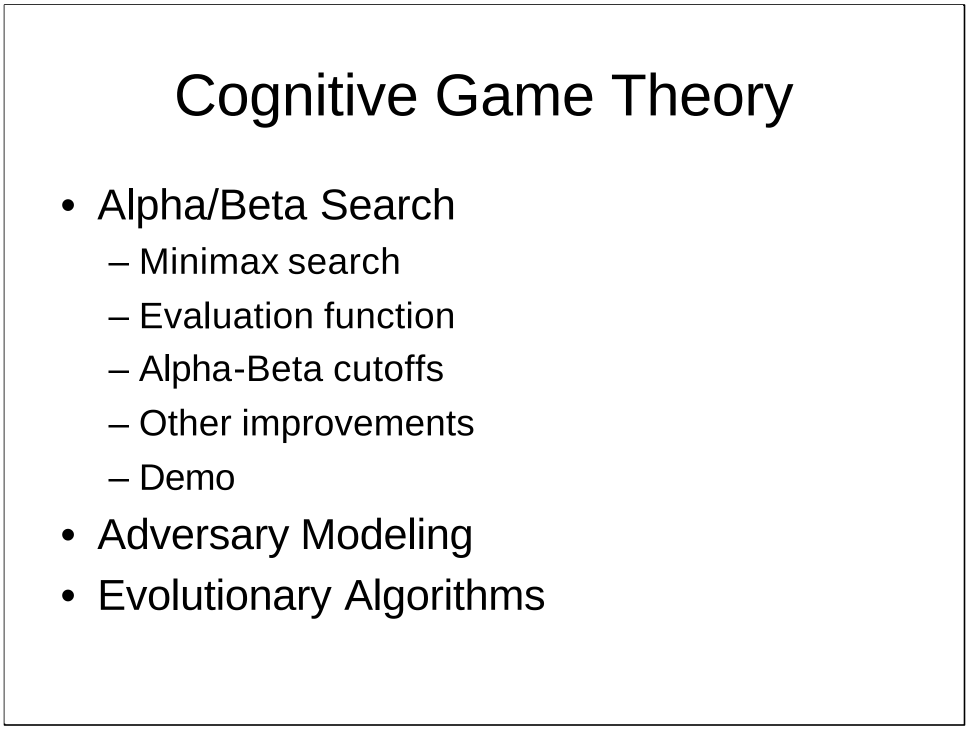
	Depth	Minimax	Alpha-Beta	+ Move ordering	Quiesc. search	Transpo. tables
Number of nodes	4	8484	2960	268	*	5855
	6	695547	99944	2436	170637	172742
	8	56902251	2676433	22383	2993949	3488690
Search time (sec.)	4	0	0	0	*	0
	6	9	1	0	2	2
	8	739	34	0	38	46



Statistics: conclusions

	First move		Jumps available	
Depth 8	Basic minimax	Advanced algorithm	Basic minimax	Advanced algorithm
Number of nodes	15237252	4835	56902251	6648
Search time (sec.)	201	0	739	0

Gain of more than 99.9% both in time and number of nodes



Cognitive Game Theory

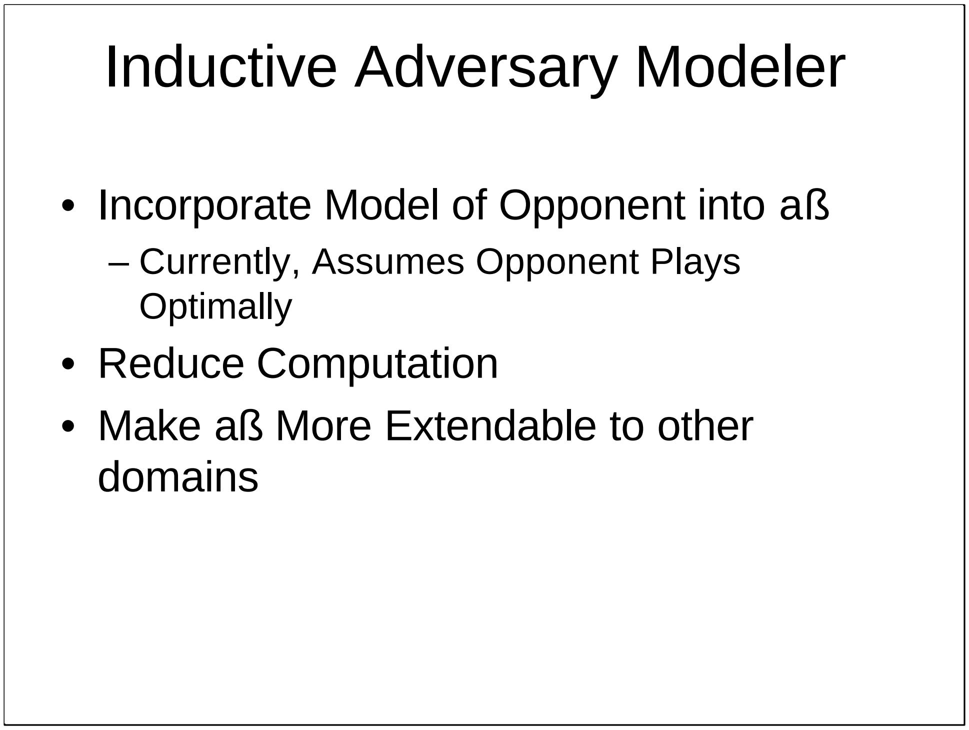
- Alpha/Beta Search
  - Minimax search
  - Evaluation function
  - Alpha-Beta cutoffs
  - Other improvements
  - Demo
- Adversary Modeling
- Evolutionary Algorithms

We return to the outline to note that the next section of this talk will now focus on a still small, but more detailed and less abstract example of how evolutionary algorithms may be applied to create chess players. This example can be found in the paper:

Kendall and Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics*, Proc. 2001 IEEE Congress on Evolutionary Computation.

Cognitive Game Theory

- Alpha/Beta Search
- Adversary Modeling
  - – Psychological Background
  - – Structure of IAM
    - Getting Chunks
    - Applying Chunks
  - – Results/Application to αβ min-max
  - – Flexibility in Other Domains
- Evolutionary Algorithms

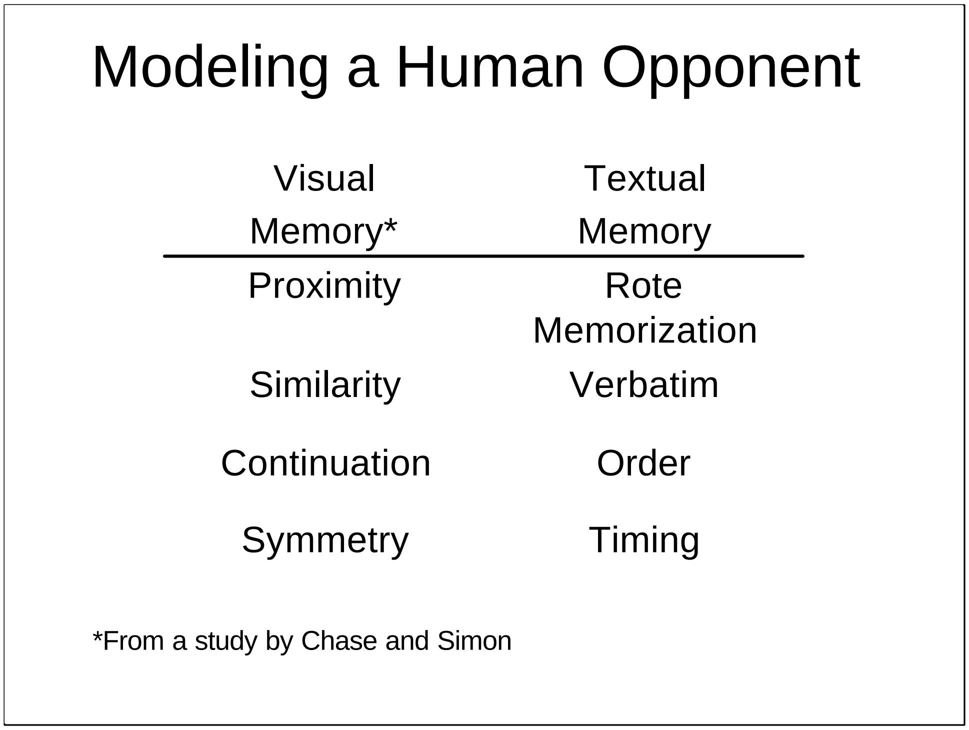


Inductive Adversary Modeler

- Incorporate Model of Opponent into aß
  - Currently, Assumes Opponent Plays Optimally
- Reduce Computation
- Make aß More Extendable to other domains

Cognitive Game Theory

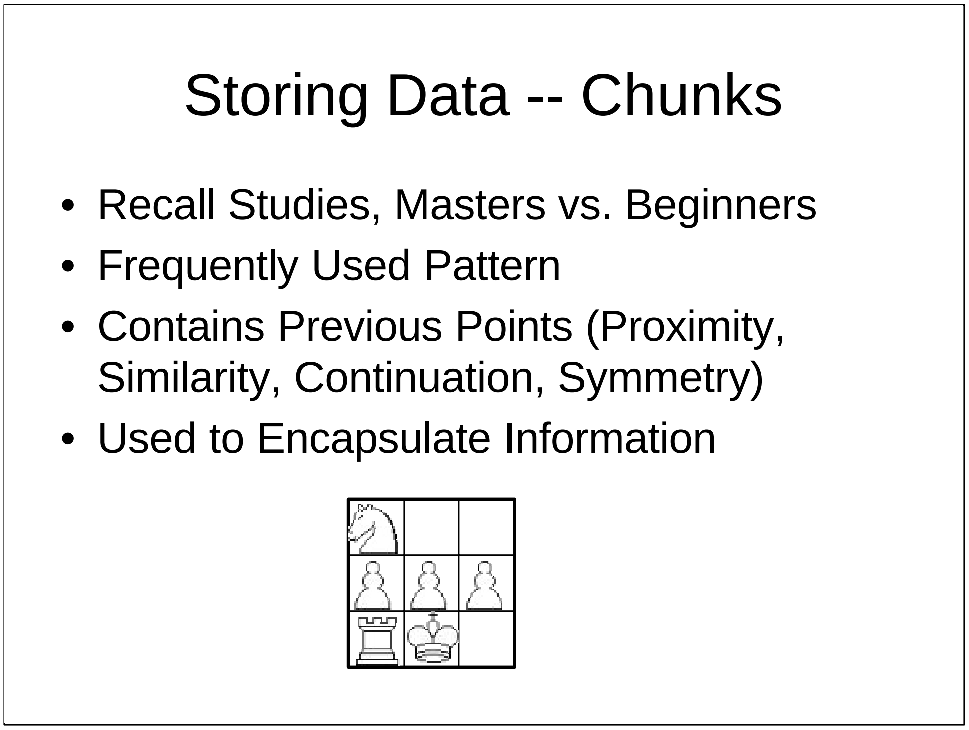
- Alpha/Beta Search
- Adversary Modeling
  - Psychological Background
  - Structure of IAM
    - Getting Chunks
    - Applying Chunks
  - Results/Application to αβ min-max
  - Flexibility in Other Domains
- Evolutionary Algorithms



Modeling a Human Opponent

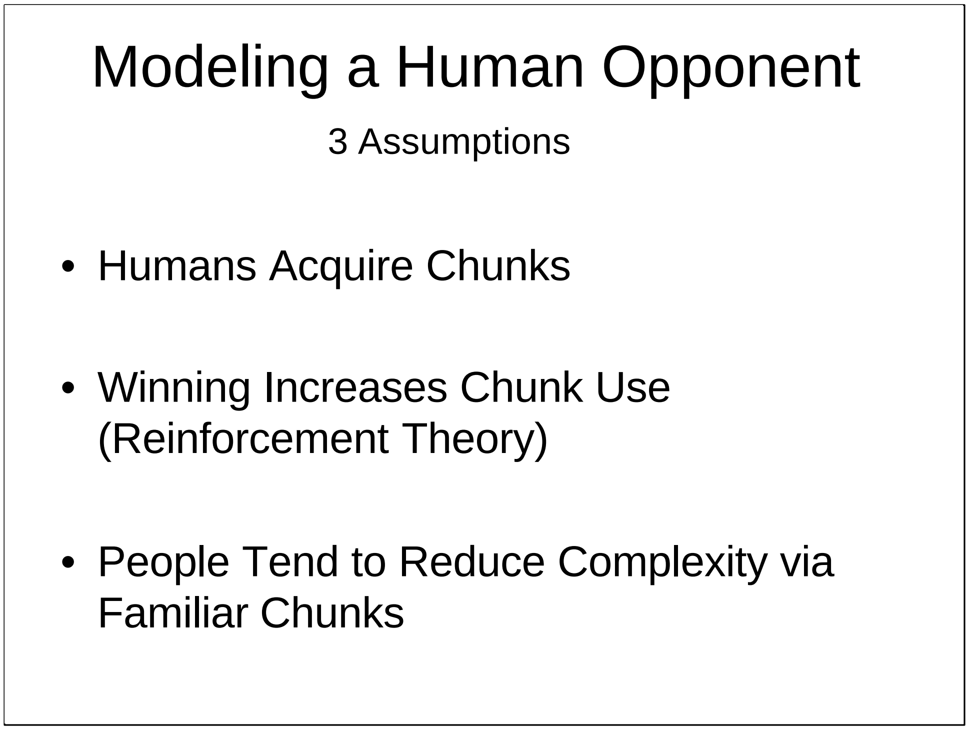
Visual Memory*	Textual Memory
Proximity	Rote Memorization
Similarity	Verbatim
Continuation	Order
Symmetry	Timing

*From a study by Chase and Simon



Storing Data -- Chunks

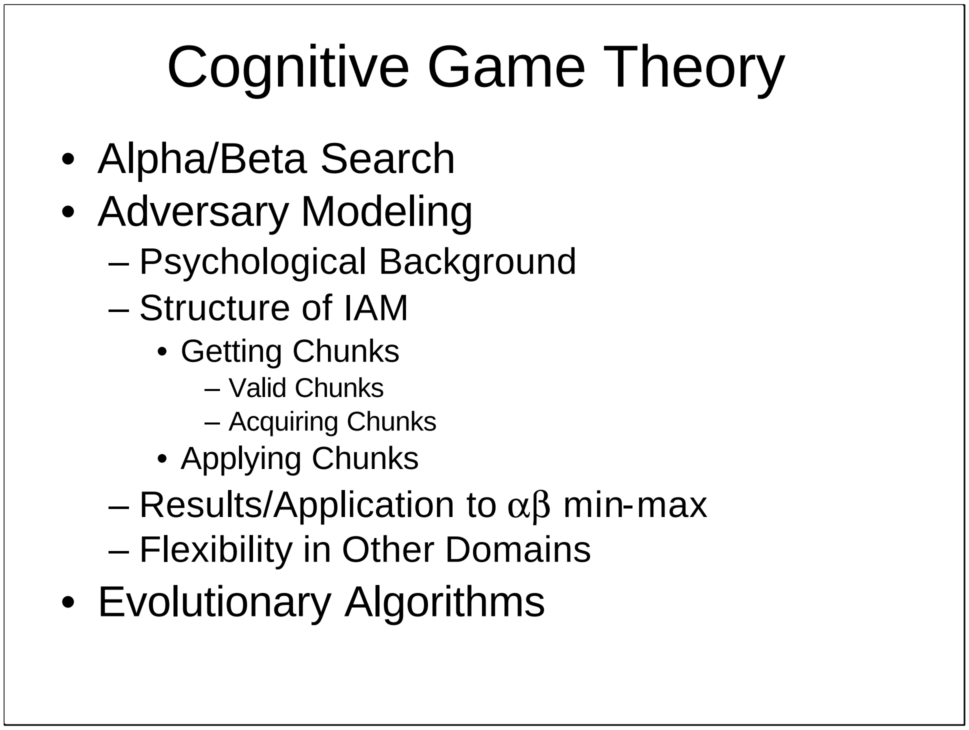
- Recall Studies, Masters vs. Beginners
- Frequently Used Pattern
- Contains Previous Points (Proximity, Similarity, Continuation, Symmetry)
- Used to Encapsulate Information



Modeling a Human Opponent

3 Assumptions

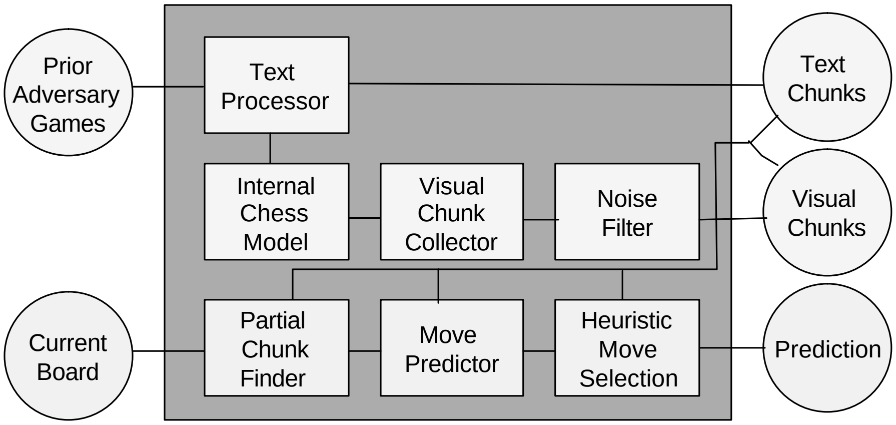
- Humans Acquire Chunks
- Winning Increases Chunk Use (Reinforcement Theory)
- People Tend to Reduce Complexity via Familiar Chunks



Cognitive Game Theory

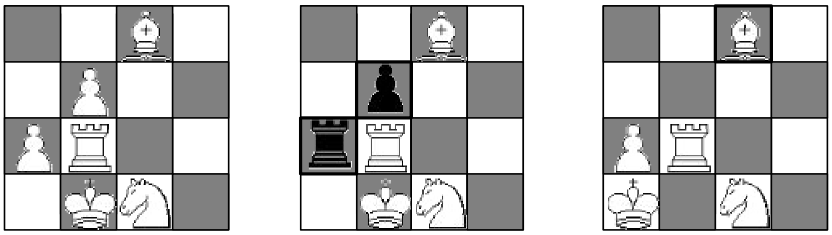
- Alpha/Beta Search
- Adversary Modeling
  - Psychological Background
  - Structure of IAM
    - Getting Chunks
      - Valid Chunks
      - Acquiring Chunks
    - Applying Chunks
  - Results/Application to $\alpha\beta$ min-max
  - Flexibility in Other Domains
- Evolutionary Algorithms

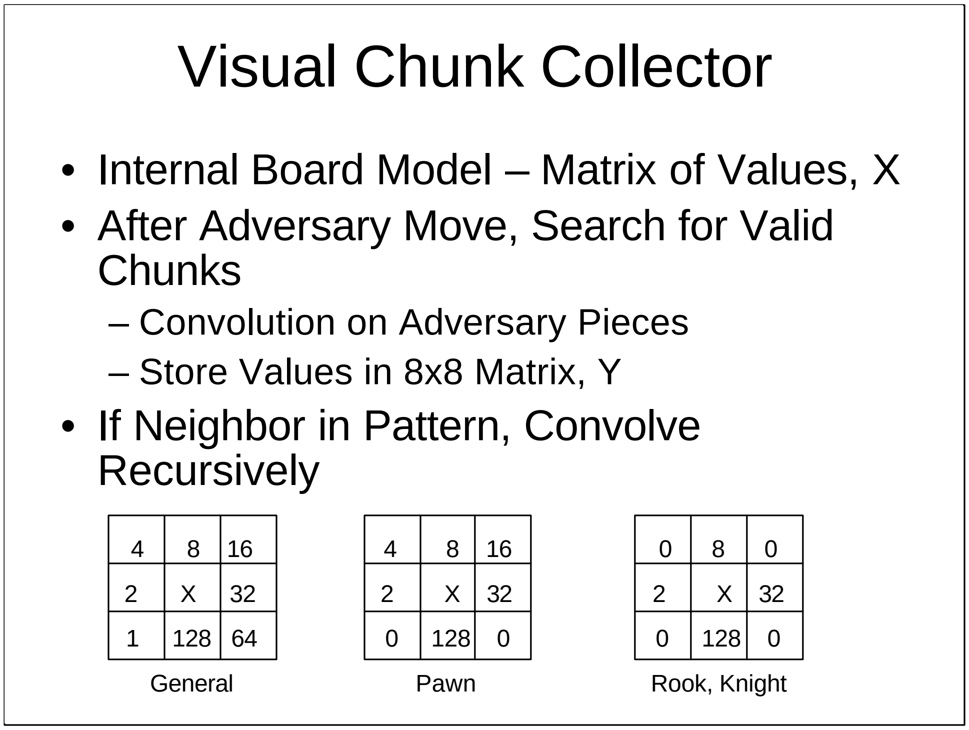
Structure of IAM



Valid Visual Chunks

- Proximity - 4x4 grid, adjacent vertically or horizontally
- Similarity - same color (exception – pawn structure)
- Continuation - pieces defending each other included
- Symmetry – symmetrical chunks stored as one
  (reduces stored chunks by about 60%)





Visual Chunk Collector

- Internal Board Model – Matrix of Values, X
- After Adversary Move, Search for Valid Chunks
  - Convolution on Adversary Pieces
  - Store Values in 8x8 Matrix, Y
- If Neighbor in Pattern, Convolve Recursively

4	8	16
2	X	32
1	128	64

General

4	8	16
2	X	32
0	128	0

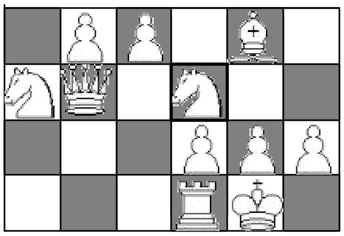
Pawn

0	8	0
2	X	32
0	128	0

Rook, Knight

Convolution Example

X:

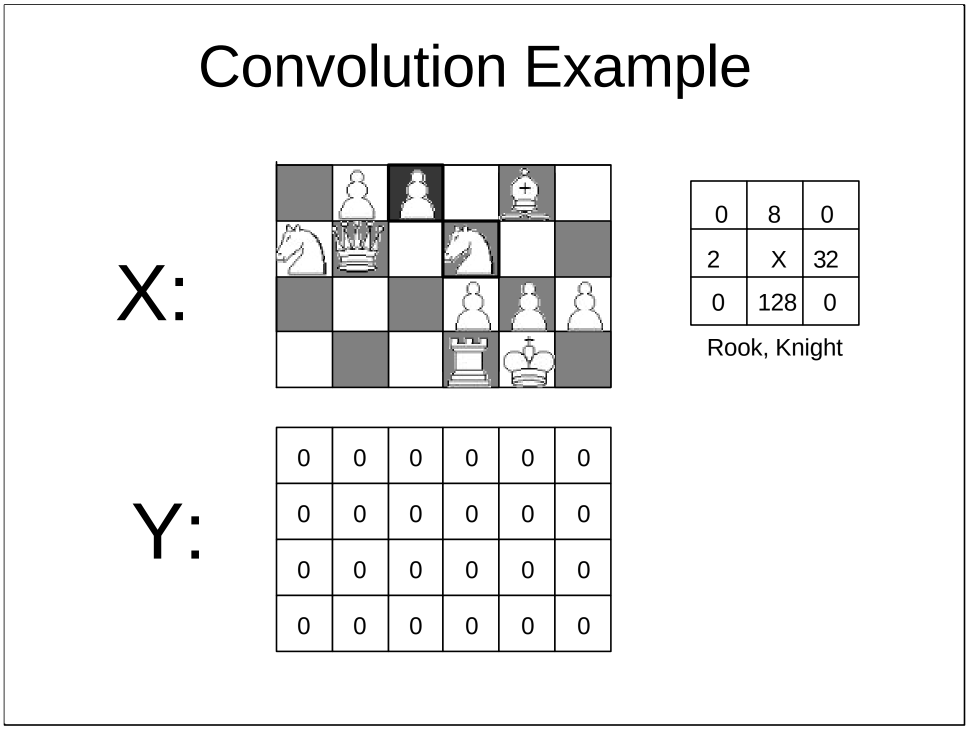


0	8	0
2	X	32
0	128	0

Rook, Knight

Y:

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0



Convolution Example

X:

0	8	0
2	X	32
0	128	0

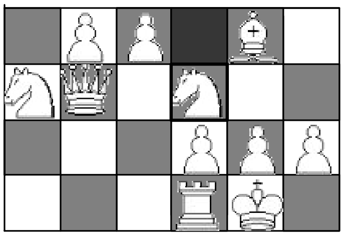
Rook, Knight

Y:

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Convolution Example

X:



0	8	0
2	X	32
0	128	0

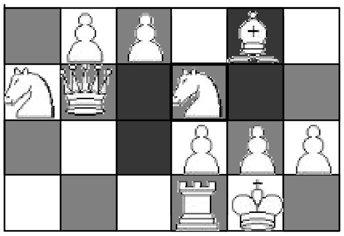
Rook, Knight

Y:

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

Convolution Example

X:

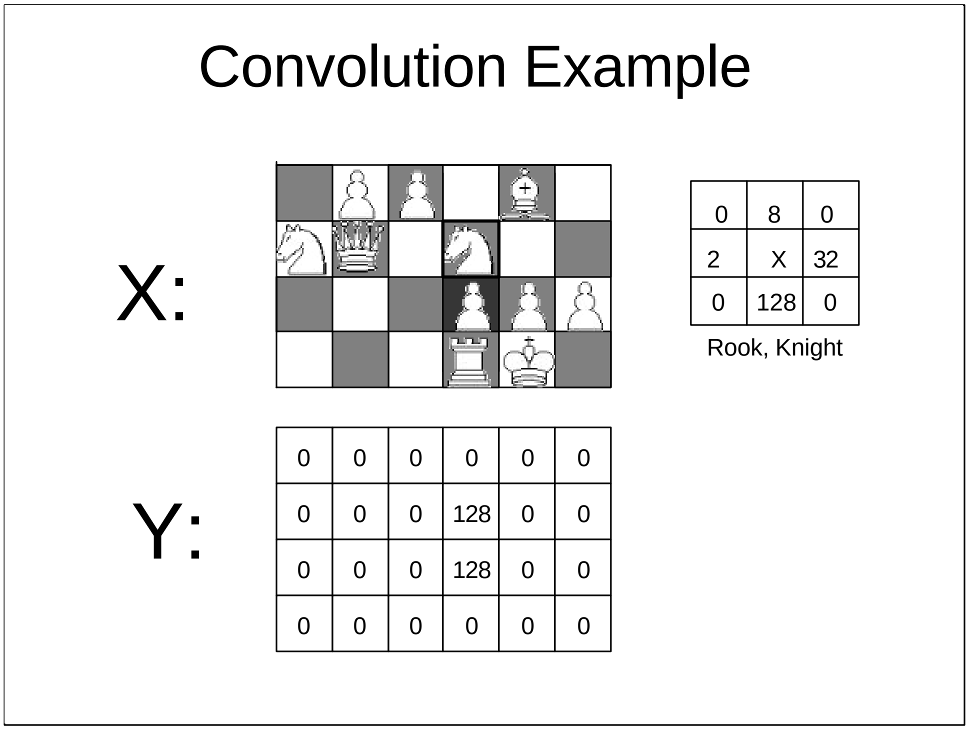


0	8	0
2	X	32
0	128	0

Rook, Knight

Y:

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0



Convolution Example

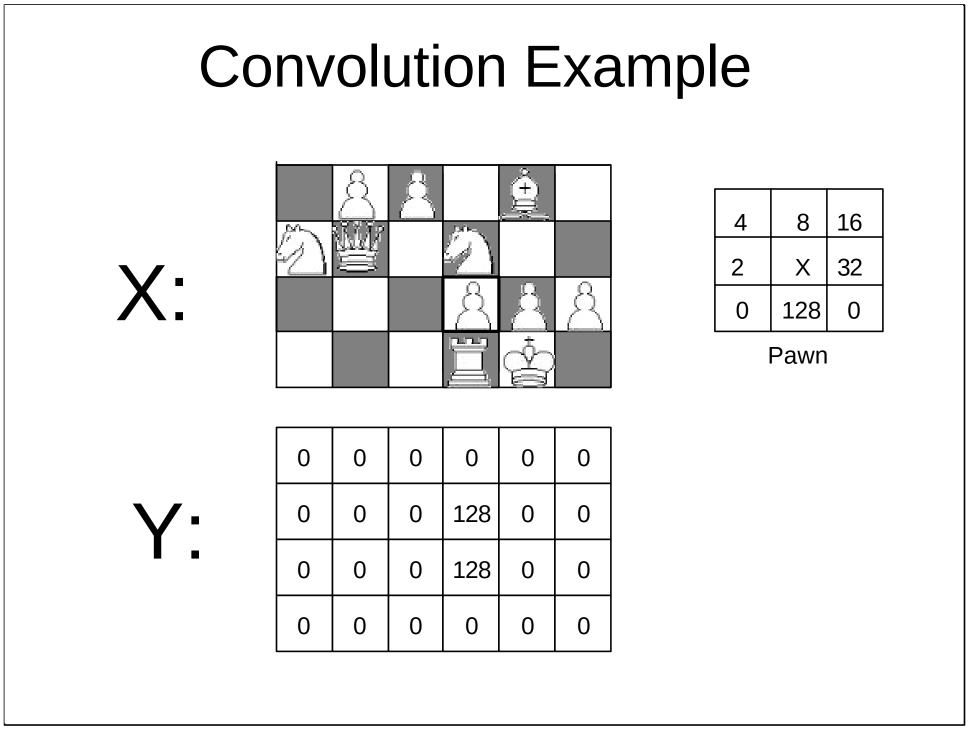
X:

0	8	0
2	X	32
0	128	0

Rook, Knight

Y:

0	0	0	0	0	0
0	0	0	128	0	0
0	0	0	128	0	0
0	0	0	0	0	0



Convolution Example

X:

4	8	16
2	X	32
0	128	0

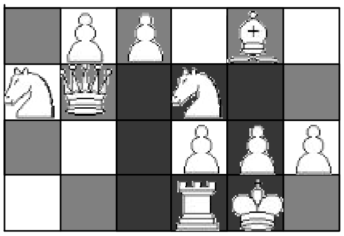
Pawn

Y:

0	0	0	0	0	0
0	0	0	128	0	0
0	0	0	128	0	0
0	0	0	0	0	0

Convolution Example

X:



4	8	16
2	X	32
0	128	0

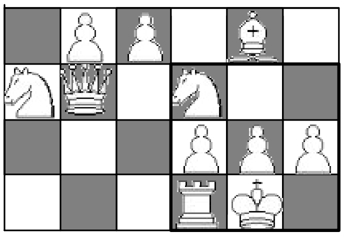
Pawn

Y:

0	0	0	0	0	0
0	0	0	136	0	0
0	0	0	196	32	0
0	0	0	128	0	0

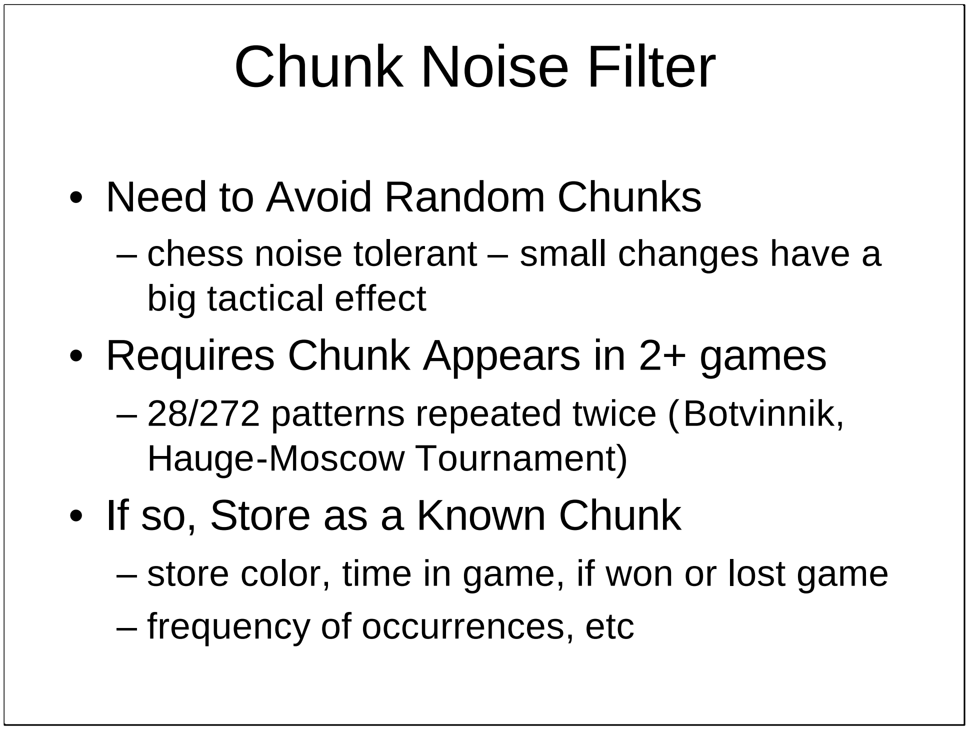
Convolution Example

X:



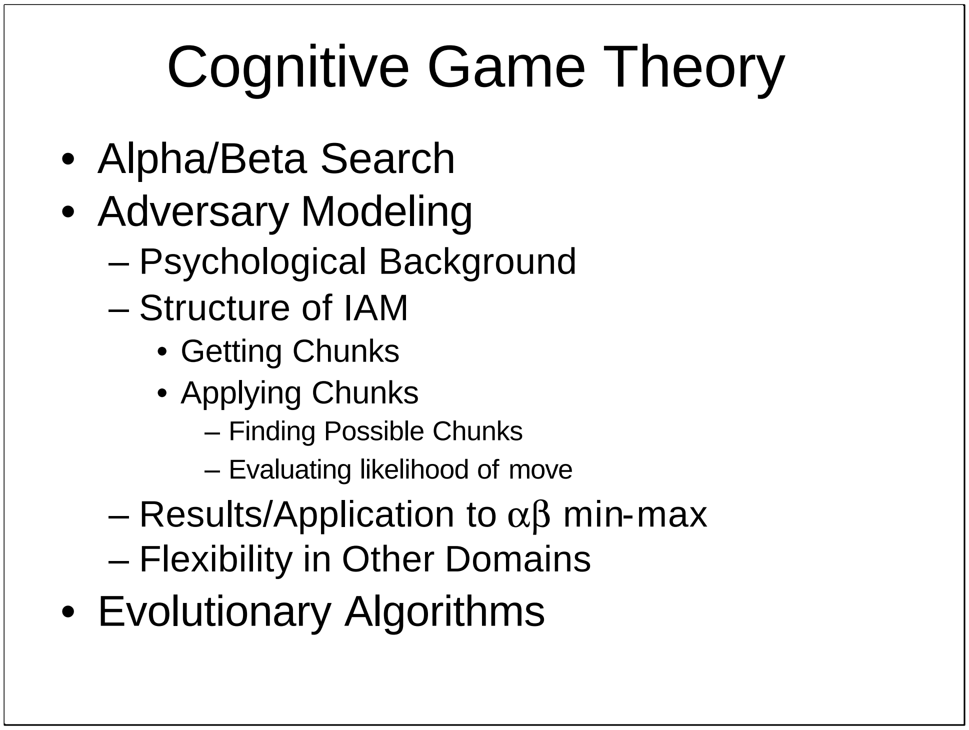
Y:

0	0	0	0	0	0
0	0	0	140	0	0
0	0	0	202	208	50
0	0	0	130	158	0



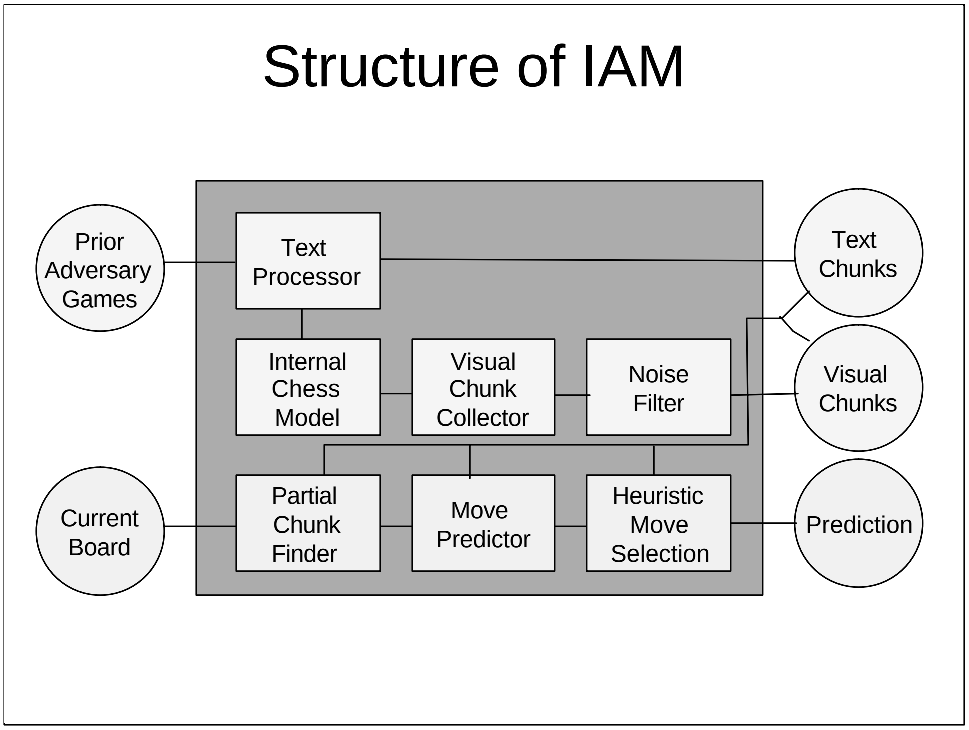
Chunk Noise Filter

- Need to Avoid Random Chunks
  - – chess noise tolerant – small changes have a big tactical effect
- Requires Chunk Appears in 2+ games
  - – 28/272 patterns repeated twice (Botvinnik, Hauge-Moscow Tournament)
- If so, Store as a Known Chunk
  - – store color, time in game, if won or lost game
  - – frequency of occurrences, etc



Cognitive Game Theory

- Alpha/Beta Search
- Adversary Modeling
  - – Psychological Background
  - – Structure of IAM
    - Getting Chunks
    - Applying Chunks
      - – Finding Possible Chunks
      - – Evaluating likelihood of move
  - – Results/Application to αβ min-max
  - – Flexibility in Other Domains
- Evolutionary Algorithms



Structure of IAM

Prior Adversary Games

Text Processor

Text Chunks

Internal Chess Model

Visual Chunk Collector

Noise Filter

Visual Chunks

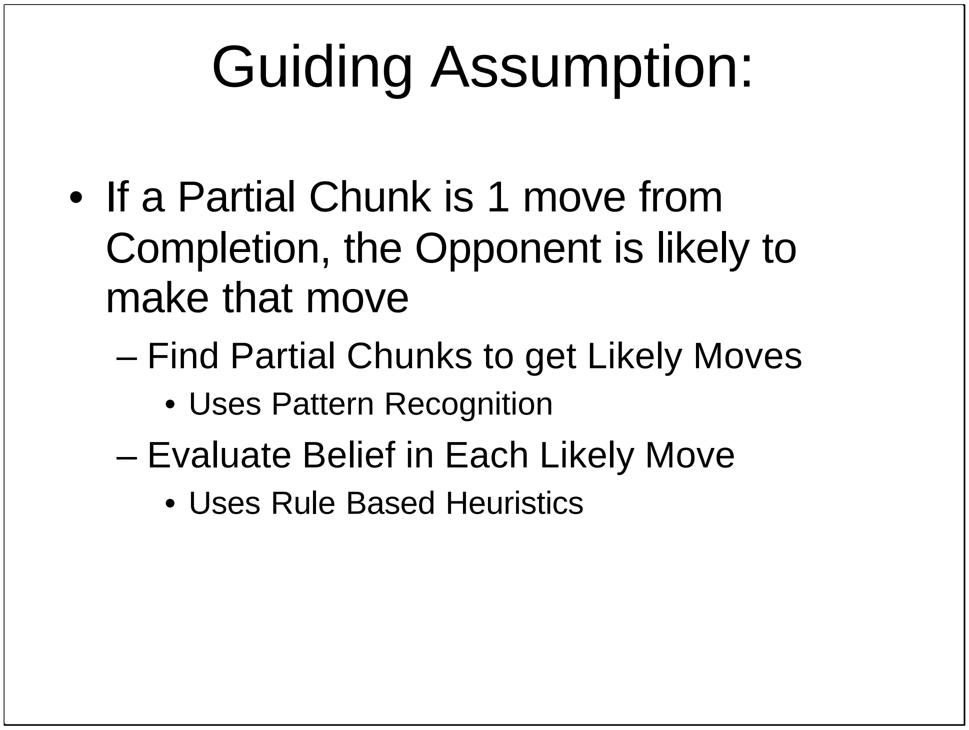
Current Board

Partial Chunk Finder

Move Predictor

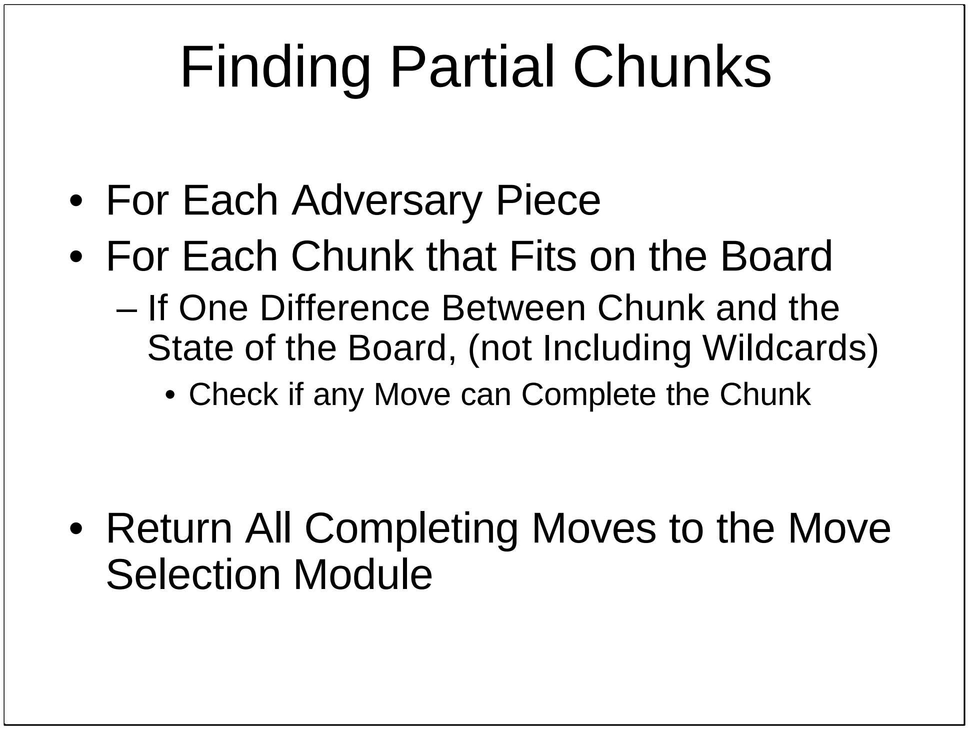
Heuristic Move Selection

Prediction



Guiding Assumption:

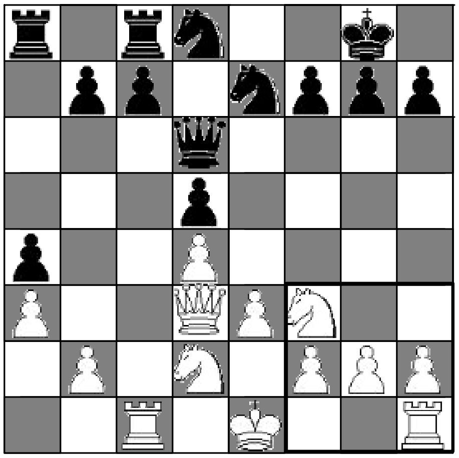
- If a Partial Chunk is 1 move from Completion, the Opponent is likely to make that move
  - – Find Partial Chunks to get Likely Moves
    - Uses Pattern Recognition
  - – Evaluate Belief in Each Likely Move
    - Uses Rule Based Heuristics

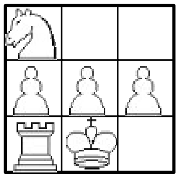


Finding Partial Chunks

- For Each Adversary Piece
- For Each Chunk that Fits on the Board
  - – If One Difference Between Chunk and the State of the Board, (not Including Wildcards)
    - Check if any Move can Complete the Chunk
- Return All Completing Moves to the Move Selection Module

Example

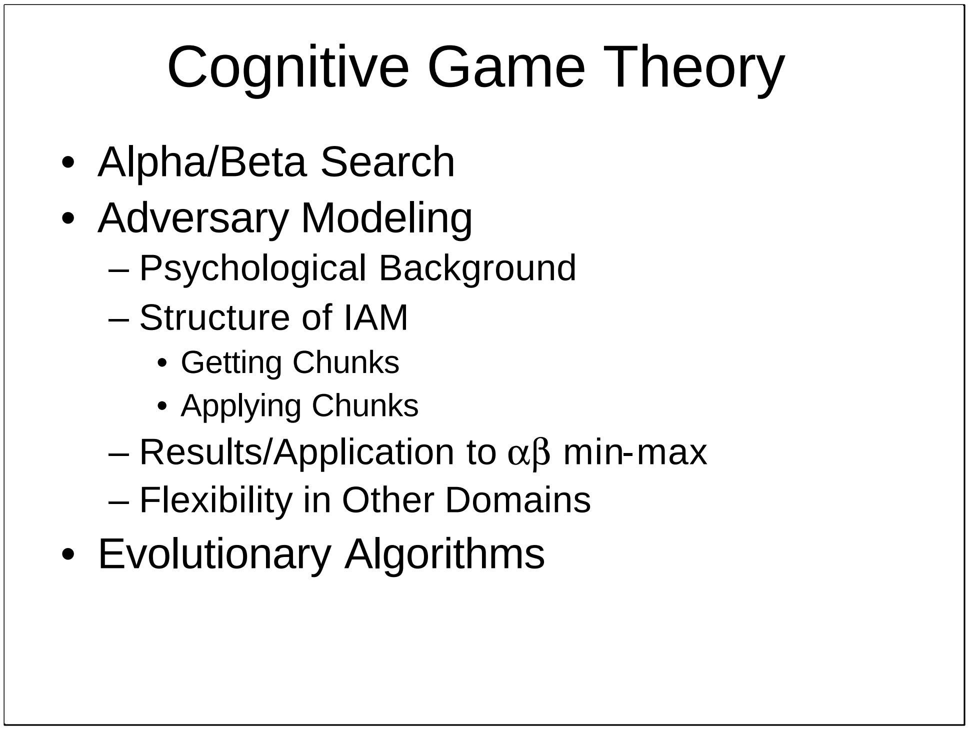




Prediction

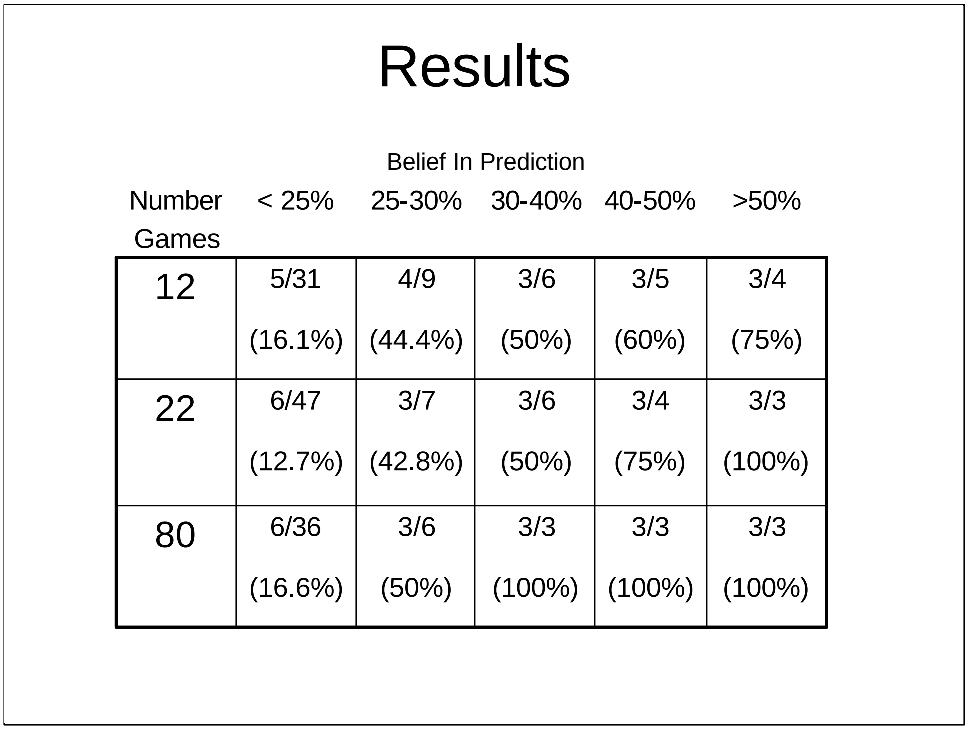
Heuristic Move Selection

- Rule Based Heuristic Algorithm
- Gives a Measure of Belief in Each Move
- Initial Belief = Frequency of Chunk
- Each Heuristic Adds/Subtracts
- Examples:
    - Favor Large Patterns
    - Favor Major Pieces
    - Favor Temporal Similarity
    - Eliminate Move if Adversary just dissolved this pattern
    - Favor Winning Patterns



Cognitive Game Theory

- Alpha/Beta Search
- Adversary Modeling
  - – Psychological Background
  - – Structure of IAM
    - Getting Chunks
    - Applying Chunks
  - – Results/Application to αβ min-max
  - – Flexibility in Other Domains
- Evolutionary Algorithms



Results

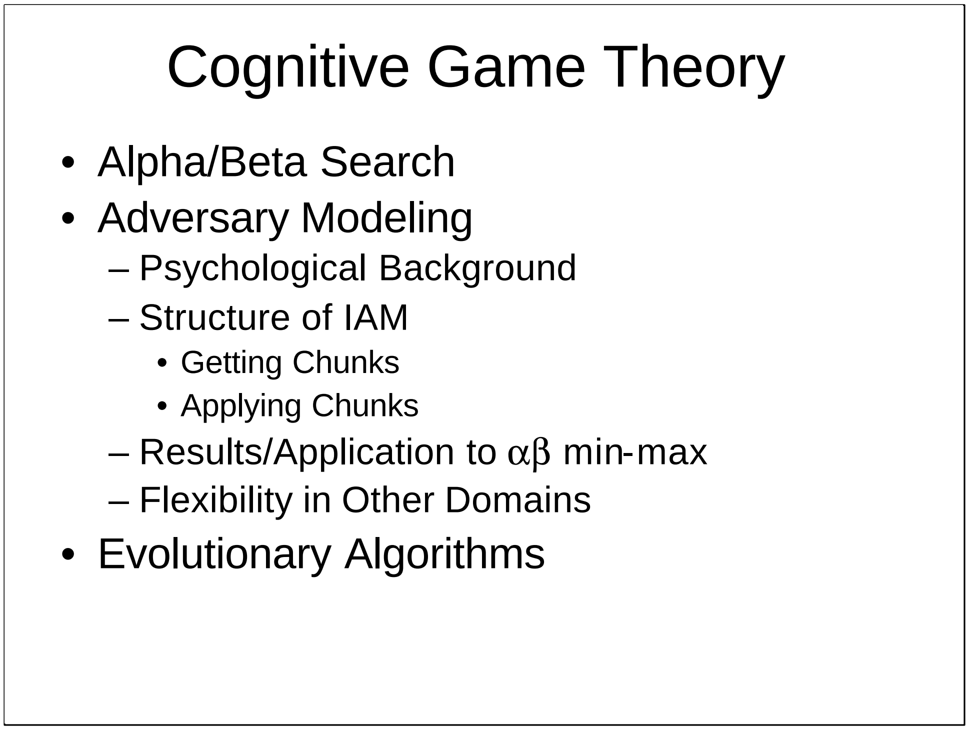
Belief In Prediction

Number Games	< 25%	25-30%	30-40%	40-50%	>50%
12	5/31 (16.1%)	4/9 (44.4%)	3/6 (50%)	3/5 (60%)	3/4 (75%)
22	6/47 (12.7%)	3/7 (42.8%)	3/6 (50%)	3/4 (75%)	3/3 (100%)
80	6/36 (16.6%)	3/6 (50%)	3/3 (100%)	3/3 (100%)	3/3 (100%)

Results -- αβ Min-Max

- Used to Prune Search Tree
  - – Develop Tree Along More Likely Moves

- Average Ply Increase – 12.5%



Cognitive Game Theory

- Alpha/Beta Search
- Adversary Modeling
  - – Psychological Background
  - – Structure of IAM
    - Getting Chunks
    - Applying Chunks
  - – Results/Application to αβ min-max
  - – Flexibility in Other Domains
- Evolutionary Algorithms

Flexibility in Other Domains

- Applicable to Other Domains
  - – Requires Competition, Adversary
  - – Military, Corporate, and Game Tactics

- Requires a Reworking of Visual Chunk Convolution Templates

Cognitive Game Theory

- Alpha/Beta Search
- Adversary Modeling
- Evolutionary Algorithms
  - Intro to Evolutionary Methodology
  - Small Example – Kendall/Whitwell
  - Evochess – Massively distributed computation for chess evolution

Evolutionary/Genetic Programs

- Create smarter agents through mutation and crossover

Mutation: "Random" change to a set of program statements

Crossover: Swapping of statements between players

- Applications in innumerable fields:
  - Optimization of Manufacturing Processes
  - Optimization of Logic Board Design
  - Machine Learning for Path Planning/Scientific Autonomy
  - CHESS!!! ☺

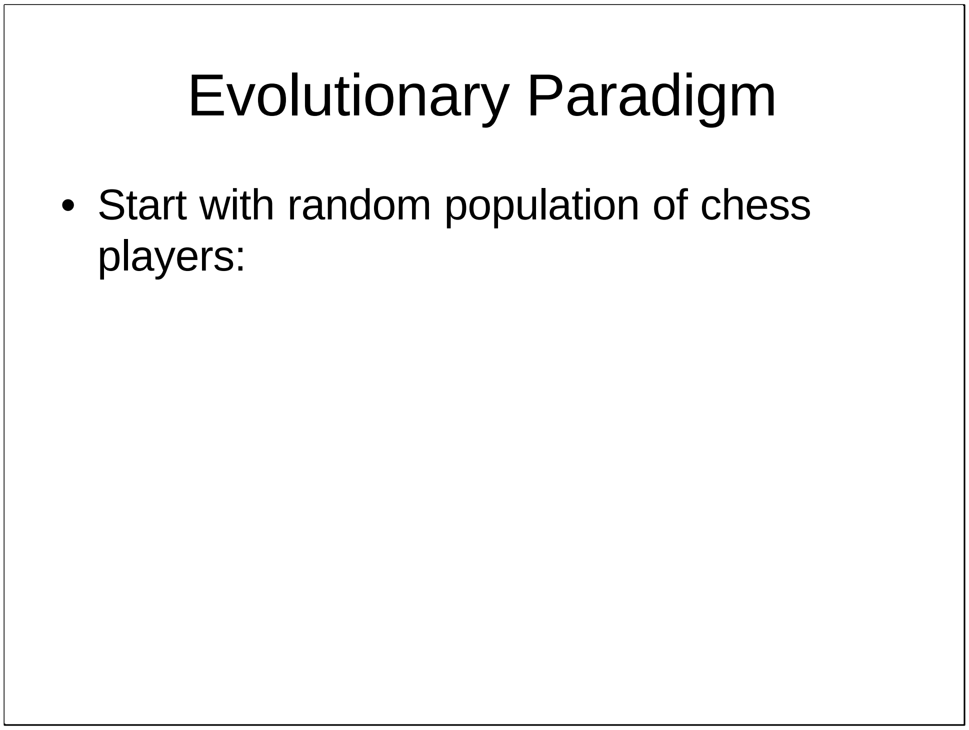
Starting from randomly created and very weak programs, evolutionary algorithms seek to create stronger or smarter programs by mimicking the principles of natural selection and of general biology. Weak programs are forced to compete with one another at a specified task. The losers are destroyed while the winners are retained. In place of the losers, modified copies of the winners are also created. These copies are created from the originals either through mutations (a random change or changes in the program's statements or structure) or through crossover (a transfer of information between two "strong" programs with the objective of discovering an even better combination of information).

Such programming techniques have been used frequently in fields such as manufacturing, circuit-board design, and of course, chess. For more information on other applications of genetic programming and evolutionary algorithms, feel free to consult:

Kojima, et. al. *An Evolutionary Algorithm Extended by Ecological Analogy to the Game of Go.* Proceedings 15 Intl. Joint Conf. on AI, 1997.

Koza. *Genetic Programming*. Encyclopedia of Computer Science and Technology. 1997.

Zbigniew. *Evolutionary Algorithms for Engineering Applications*. 1997.



Evolutionary Paradigm

- Start with random population of chess players:

Let us walk through a simple, abstracted example to illustrate concretely the methodology we will be using.

In the slide above, there are six randomly created blobs. Some blobs are already smarter chess players than the others simply by random luck. (For example the blob in the lower right corner doesn’t even have eyes, so probably will not be as good at chess as the others!)

These six blobs may represent entire chess-playing programs (as in the case of the EvoChess project) or merely a certain portion of such a program (as in the Kendall/Whitwell evaluation function example we’ll see in a moment).

Evolutionary Paradigm

- Population plays games against each other:

We take this original population and allow it to compete. For example, we could have each blob play a best of 3 chess match against every other blob. Or (as is illustrated above) we could perform pairwise comparisons between the various blobs. Each pair of blobs could play a best of 3 match and the results would be recorded.

Evolutionary Paradigm

- Losers are killed and removed from population:

Those blobs who lost two of three games would then be culled from the population (as demonstrated by the blood splats above). The assumption here being that these weaker blobs represent areas of the space which are no longer productive to explore. The stronger blobs, on the other hand, represent areas of the space which may yield even stronger players if we continue to explore players similar to them.

Evolutionary Paradigm

- Winners mate and have (possibly mutated) offspring:

Pure
mutation

To maintain the population size and also to ensure that we are not only exploiting the space, but also exploring it, we use the biological models of mutations and crossovers to create new chess players starting from the features of the strongest blobs remaining in the population.

The slide above demonstrates that sometimes these newly-created blobs can move closer to the optimal chess player (as denoted by the pictures of myself and Professor Brian Williams) and that sometimes the newly-created blobs can be de-evolved versions of the former player, moving farther from the desired intelligent agent and becoming something much worse.

Evolutionary Paradigm

- New Population Competes:

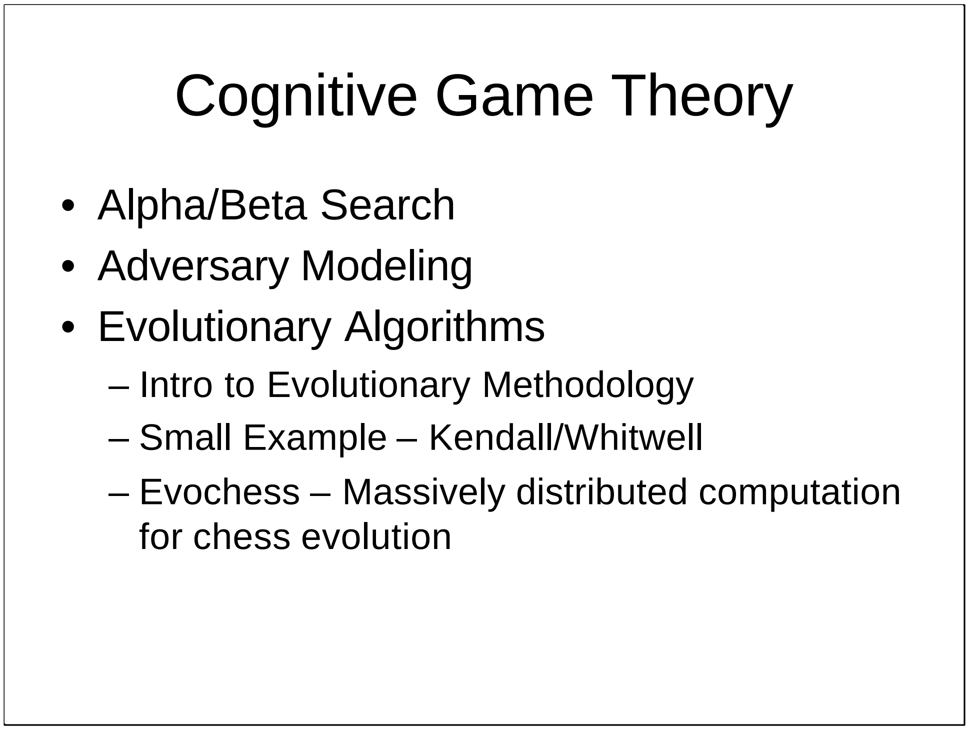
The new population, which at worst contains algorithms of the same strength as the previous generation and at best contains algorithms of better strength, is again allowed to compete. In the example above, it is discovered that Professor Williams is a more capable chess player than either myself, Bill Gates, or the three remaining blobs. Who knew?

Evolutionary Paradigm

- Eventually the population converges, mutations become reduced, and the whole population converges:

As time goes by, mutations are allowed to become less and less severe or frequent. This drives the population to converge. When the differences between the population's performance become slight or when some other relevant stopping criteria has been met, the best player within the population is declared the "most evolved" player. This evolved player is usually much stronger than the original weak blobs that were started with, and yet minimal domain-specific knowledge has been required of the programmer. Using this technique, a programmer could easily develop a chess program that not only had greater computational search resources available to it, but could also conceptually understand the game better than the programmer himself. Such a result is extremely intriguing and useful in many real-world applications.

In the case above, we see that eventually our population of blobs has converged to become the great Gary Kasparov. Now THAT'S a powerful algorithm.

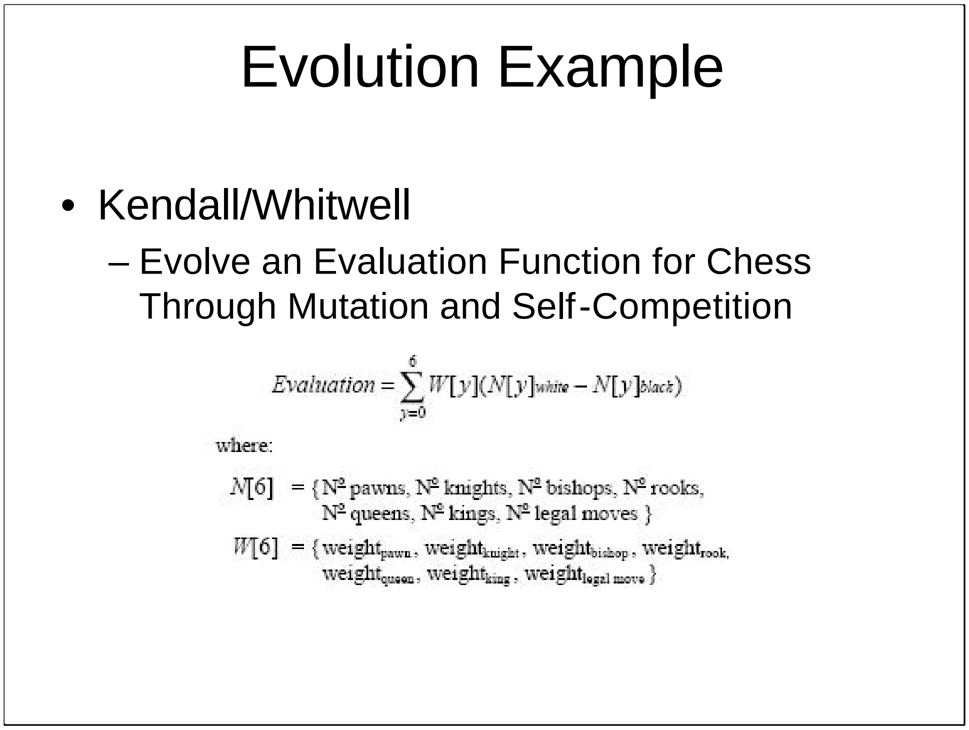


Cognitive Game Theory

- Alpha/Beta Search
- Adversary Modeling
- Evolutionary Algorithms
  - – Intro to Evolutionary Methodology
  - – Small Example – Kendall/Whitwell
  - – Evochess – Massively distributed computation for chess evolution

We return to the outline to note that the next section of this talk will now focus on a still small, but more detailed and less abstract example of how evolutionary algorithms may be applied to create chess players. This example can be found in the paper:

Kendall and Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics*, <u>Proc. 2001 IEEE Congress on Evolutionary Computation</u>.



Evolution Example

- Kendall/Whitwell
  - Evolve an Evaluation Function for Chess Through Mutation and Self-Competition

$$Evaluation = \sum_{y=0}^{6} W[y](N[y]_{white} - N[y]_{black})$$

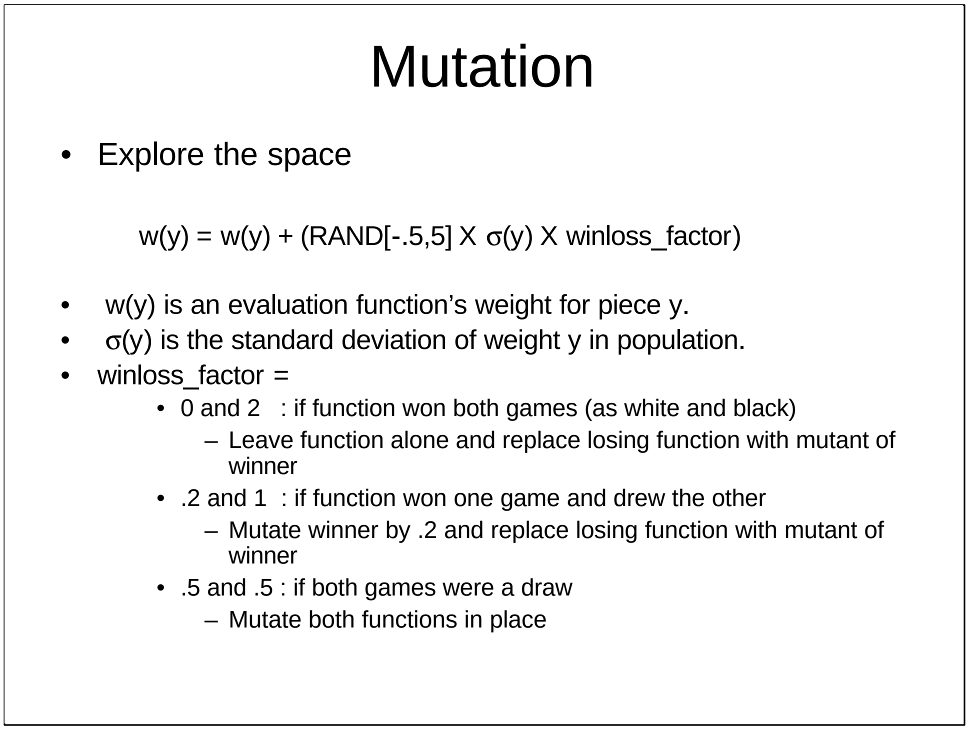
where:

$$N[6] = \{\text{Nº pawns, Nº knights, Nº bishops, Nº rooks, Nº queens, Nº kings, Nº legal moves}\}$$

$$W[6] = \{\text{weight}_{\text{pawn}}, \text{weight}_{\text{knight}}, \text{weight}_{\text{bishop}}, \text{weight}_{\text{rook,}} \text{weight}_{\text{queen}}, \text{weight}_{\text{king}}, \text{weight}_{\text{legal move}}\}$$

In this paper, the authors present a method by which an evaluation function for chess can be created. (Evaluation functions are covered earlier in the talk by Jeremie Pouly, but in brief are a method by which a computer chess player can discern how “good” or “bad” he is doing given a certain chessboard configuration.) These evaluation functions are generally the hardest part of a chess program for a programmer to create because they require the incorporation of expert knowledge which may be unavailable or very painstaking to obtain. Because of its domain-independent learning capability, evolutionary algorithms are perfectly suited to the task of evaluation function creation.

Kendall and Whitwell defined their evaluation function as a weighted sum of the difference in the numbers of each piece along with a seventh value representing the number of available moves for a given player. The authors chose to incorporate as much domain-knowledge as they had in order to limit the scope of their algorithm, but this in general would not be necessary. They could have started from an even more general function with zero chess knowledge, though the convergence to a suitable player might have been extremely time-consuming.



Mutation

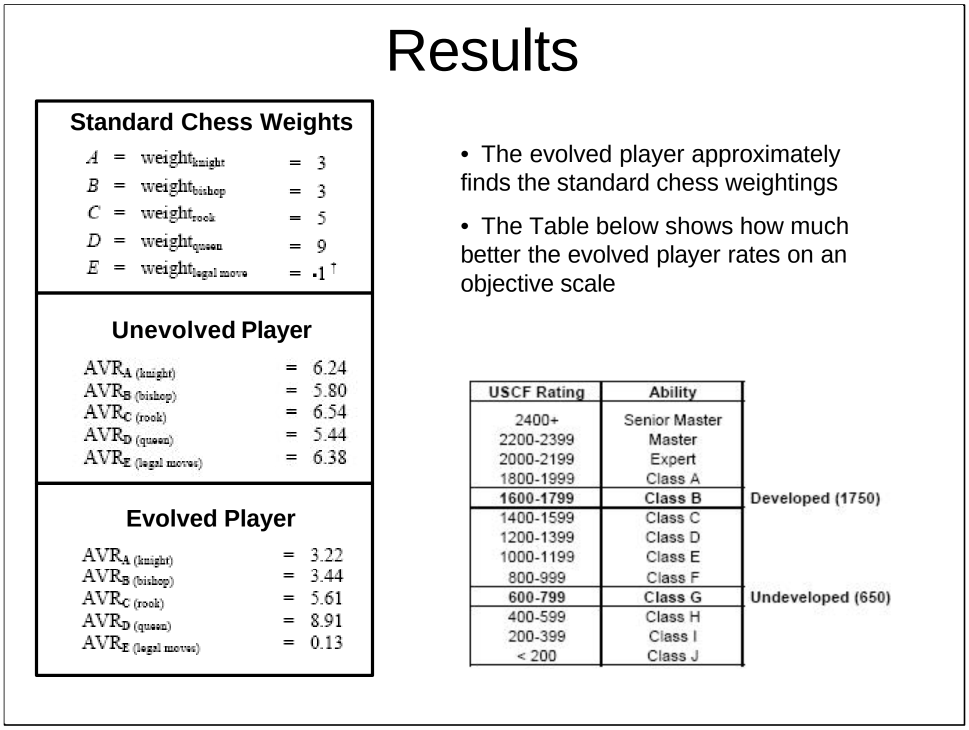
- Explore the space

$$w(y) = w(y) + (\text{RAND}[-.5,5] \text{ X } \sigma(y) \text{ X } \text{winloss\_factor})$$

- w(y) is an evaluation function's weight for piece y.
- $\sigma(y)$ is the standard deviation of weight y in population.
- winloss_factor =
    - 0 and 2 : if function won both games (as white and black)
        - Leave function alone and replace losing function with mutant of winner
    - .2 and 1 : if function won one game and drew the other
        - Mutate winner by .2 and replace losing function with mutant of winner
    - .5 and .5 : if both games were a draw
        - Mutate both functions in place

Given this general evaluation function, the authors created a random population of chess players (all of which were simple alpha-beta 3-ply searchers) and started the evolutionary process by performing competitions. After each competition the loser was completely erased from memory. In its place, a copy of the winner was placed with slight mutations.

The slide above shows the equation for performing mutations. Basically it consists of adding or subtracting a random value to the weights for a certain piece. This value is scaled depending on the outcome of the competition preceeding the mutation. If the winning algorithm won both games, one copy is left unchanged while the other is mutated by a large factor. This ensures that while we hold onto the currently strongest players in the population, we also continue to broadly search the space for better options. If the winning algorithm won one game and tied the other game, one copy is changed slightly and the second copy is changed by a moderate factor, again in the hopes of further improvement. In the case where the match was a draw, both algorithms are changed by a moderate factor and replaced in the population. The final scaling factor in the mutations, and the innovative portion of Kendall and Whitwell's work, is the standard deviation of the population. This provides an intrinsic method by which to reduce the severity of mutations as the population begins to converge. Previous methods had used an extrinsic relationship (such as a function which exponentially decreased with time) that required hand-tuning in order to obtain the proper mutation level at different stages in the evolution.



Results

Standard Chess Weights

$$A = \text{weight}_{\text{knight}} = 3$$
$$B = \text{weight}_{\text{bishop}} = 3$$
$$C = \text{weight}_{\text{rook}} = 5$$
$$D = \text{weight}_{\text{queen}} = 9$$
$$E = \text{weight}_{\text{legal move}} = .1^{\dagger}$$

Unevolved Player

$$\text{AVR}_{\text{A (knight)}} = 6.24$$
$$\text{AVR}_{\text{B (bishop)}} = 5.80$$
$$\text{AVR}_{\text{C (rook)}} = 6.54$$
$$\text{AVR}_{\text{D (queen)}} = 5.44$$
$$\text{AVR}_{\text{E (legal moves)}} = 6.38$$

Evolved Player

$$\text{AVR}_{\text{A (knight)}} = 3.22$$
$$\text{AVR}_{\text{B (bishop)}} = 3.44$$
$$\text{AVR}_{\text{C (rook)}} = 5.61$$
$$\text{AVR}_{\text{D (queen)}} = 8.91$$
$$\text{AVR}_{\text{E (legal moves)}} = 0.13$$

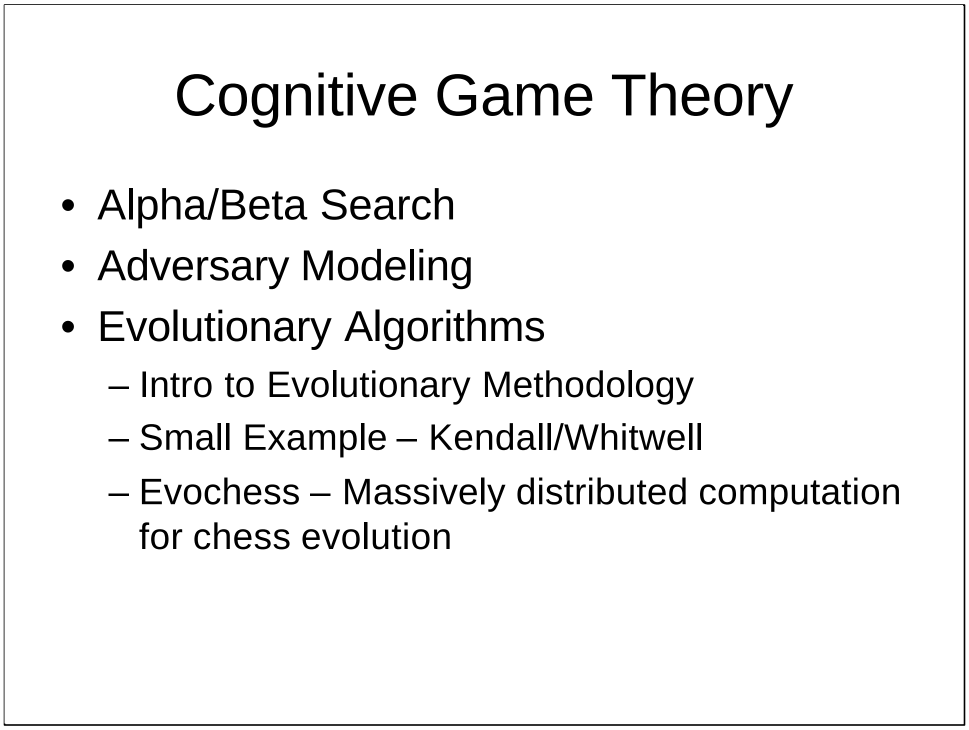
- The evolved player approximately finds the standard chess weightings
- The Table below shows how much better the evolved player rates on an objective scale

USCF Rating	Ability	
2400+	Senior Master	
2200-2399	Master	
2000-2199	Expert	
1800-1999	Class A	
1600-1799	**Class B**	**Developed (1750)**
1400-1599	Class C	
1200-1399	Class D	
1000-1199	Class E	
800-999	Class F	
600-799	**Class G**	**Undeveloped (650)**
400-599	Class H	
200-399	Class I	
< 200	Class J	

This algorithm is shown to be quite capable of creating a useful chess evaluation function. On the left, the first table represents standard weightings discovered by human experts through countless years of play. The second table represents the initial randomized weights of the population used by Kendall and Whitwell, and the bottom table represents the weights evolved using the evolutionary algorithm discussed above. It is seen that these weights closely resemble the standard chess weightings. (The authors did not conduct a study to determine if the evolved weights actually performed better than the standard human weights. This is, however, not really the point of the evolution. We would hope to apply evolution to situations in which human domain knowledge is unavailable, not improve upon existing knowledge.)

To the right a table depicts the level of the unevolved chess player based on the United States Chess Federation's rating scale. It is seen that the player with random weights performed a full five classes worse than the evolved player. The evolved player reaches a level bordering on expert.

This vivdly demonstrates that evolutionary algorithms are a powerful and viable method for creating artificial intelligence. (After all, it is likely that evolution created human intelligence, why shouldn't it be able to do the same for computers!)



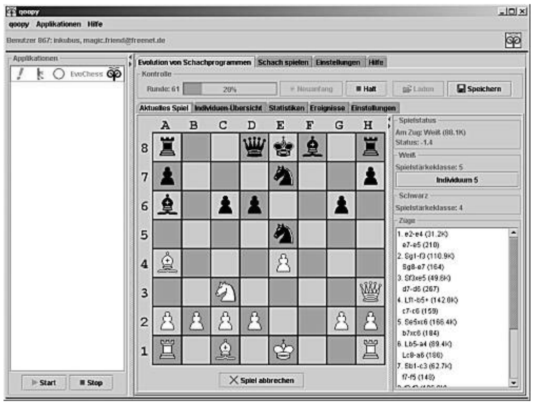
Cognitive Game Theory

- Alpha/Beta Search
- Adversary Modeling
- Evolutionary Algorithms
  - Intro to Evolutionary Methodology
  - Small Example – Kendall/Whitwell
  - Evochess – Massively distributed computation for chess evolution

We return to the outline to note that the next section of this talk will now focus on a still small, but more detailed and less abstract example of how evolutionary algorithms may be applied to create chess players. This example can be found in the paper:

Kendall and Whitwell. *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics*, Proc. 2001 IEEE Congress on Evolutionary Computation.

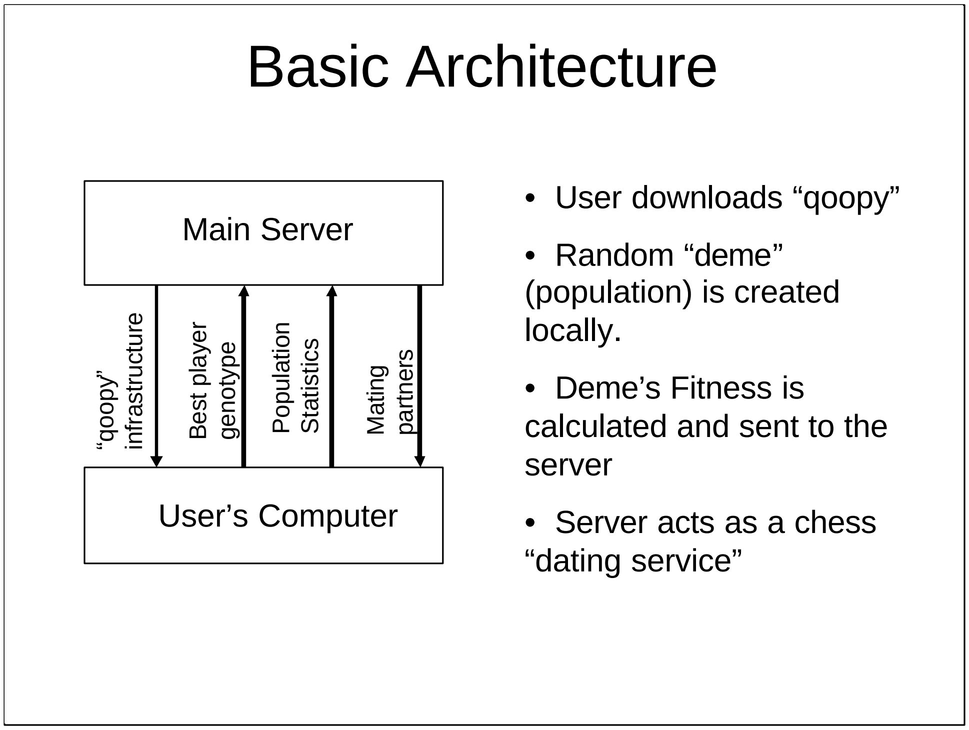
What is EvoChess?



```
Benutzer 867: inkubus, magic.friend@freenet.de
Runde: 61   20%
Am Zug: Weiß (88.1K)
Status: -1.4
Spielstärkeklasse: 5
Individuum 5
Spielstärkeklasse: 4
1. e2-e4 (31.2K)
   e7-e5 (210)
2. Sg1-f3 (110.9K)
   Sg8-e7 (164)
3. Sf3xe5 (49.6K)
   d7-d6 (267)
4. Lf1-b5+ (142.0K)
   c7-c6 (159)
5. Se5xc6 (166.4K)
   b7xc6 (184)
6. Lb5-a4 (89.4K)
   Lc8-a6 (186)
7. Sb1-c3 (62.7K)
   f7-f5 (148)
```

A distributed project to evolve better chess-playing algorithms

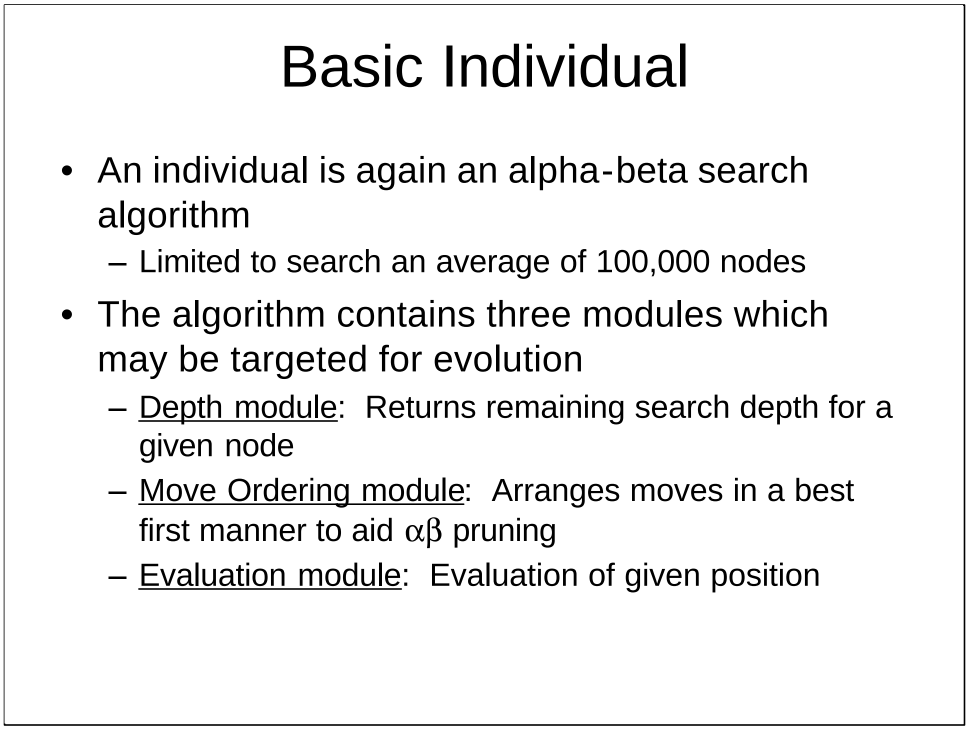
The final portion of this talk focuses on the EvoChess project developed by Gross, Albrecht, Kantschik, and Banzhaf. This project marked the first massively distributed evolution of a chess-playing program and is still arguably one of the most ambitious evolution projects ever undertaken. Like familiar SETI programs, EvoChess allowed internet users from around the world to download a chess-evolving client onto their PC and maintain a local population of evolving chess players which could be accessed by a central server which performed the necessary interactions between different users' populations.



Basic Architecture

- User downloads "qoopy"
- Random "deme" (population) is created locally.
- Deme's Fitness is calculated and sent to the server
- Server acts as a chess "dating service"

The general operation of the EvoChess algorithm is shown above. An internet user first downloaded the necessary distributed architecture known as "qoopy" onto his home computer. This program created a small population of chess-playing individuals on the local machine. These programs were allowed to compete against a number of standardized chess-playing algorithms and their fitness was calculated relative to these standard programs. Information about this population, specifically the genotype of its strongest player, were then sent back automatically to the main EvoChess server. This server then acted as a "dating service," sending the best genotypes back out to weaker populations as "mating" partners. In this way, information was transferred from population to population and on average, the entire EvoChess population began to grow smarter.

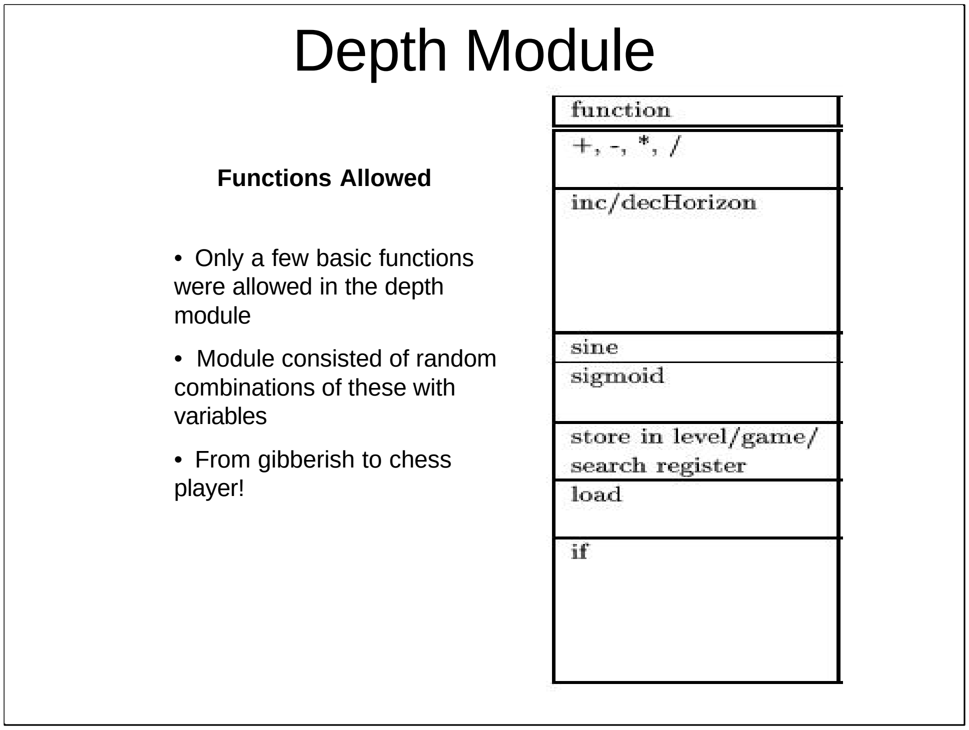


Basic Individual

- An individual is again an alpha-beta search algorithm
  - Limited to search an average of 100,000 nodes
- The algorithm contains three modules which may be targeted for evolution
  - Depth module: Returns remaining search depth for a given node
  - Move Ordering module: Arranges moves in a best first manner to aid αβ pruning
  - Evaluation module: Evaluation of given position

The basic chess-playing individual used in the EvoChess project was a very ambitious one. It consisted of a basic alpha-beta search algorithm. However, instead of being depth-limited, the algorithm was merely limited to search an average of 100,000 nodes per move.

Three main portions of the player could feel the effects of evolution. The depth module rated different branches of the search as “interesting” or “fruitless” and chose whether to spend nodes on further searching or abandon a particular alpha-beta branch. The move-ordering module selected the order in which the various possible moves available to a player at a given time were explored. An intelligent ordering of moves can greatly decrease the size of an alpha-beta tree by allowing earlier pruning. Finally, like in the Kendall and Whitwell example, an evaluation module was needed which could determine how “good” or “bad” a certain board configuration appeared to a player. EvoChess’s evaluation function, however, contained over twenty different parameters of interest, making it a great deal more complicated than the early authors’ function.



Depth Module

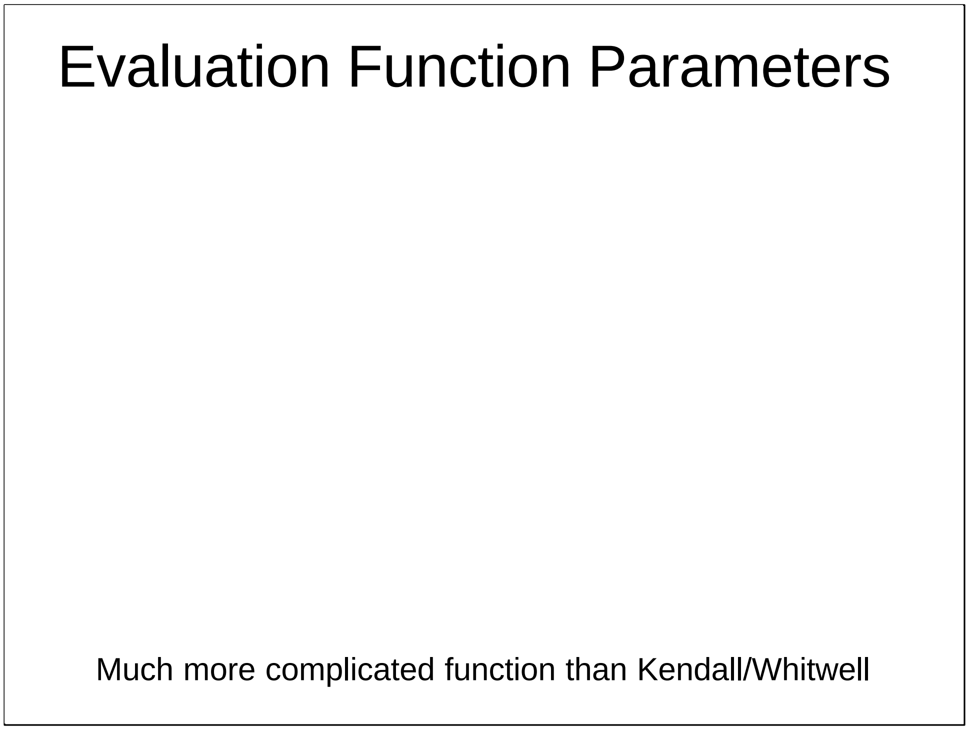
Functions Allowed

- Only a few basic functions were allowed in the depth module
- Module consisted of random combinations of these with variables
- From gibberish to chess player!

function
+, -, *, /
inc/decHorizon
sine
sigmoid
store in level/game/ search register
load
if

One of the most interesting and ambitious portions of the player was the depth-module. This module relied on genetic programming techniques. It originally began as a random arrangement of functions and literals from a finite set (the functions available are depicted in the table above.) The initial depth modules of first generation players were therefore usually just complete gibberish. While most authors usually attempt to speed convergence by applying domain knowledge to limit the scope of a program, EvoChess used an absolute minimum of information in its original random depth modules.

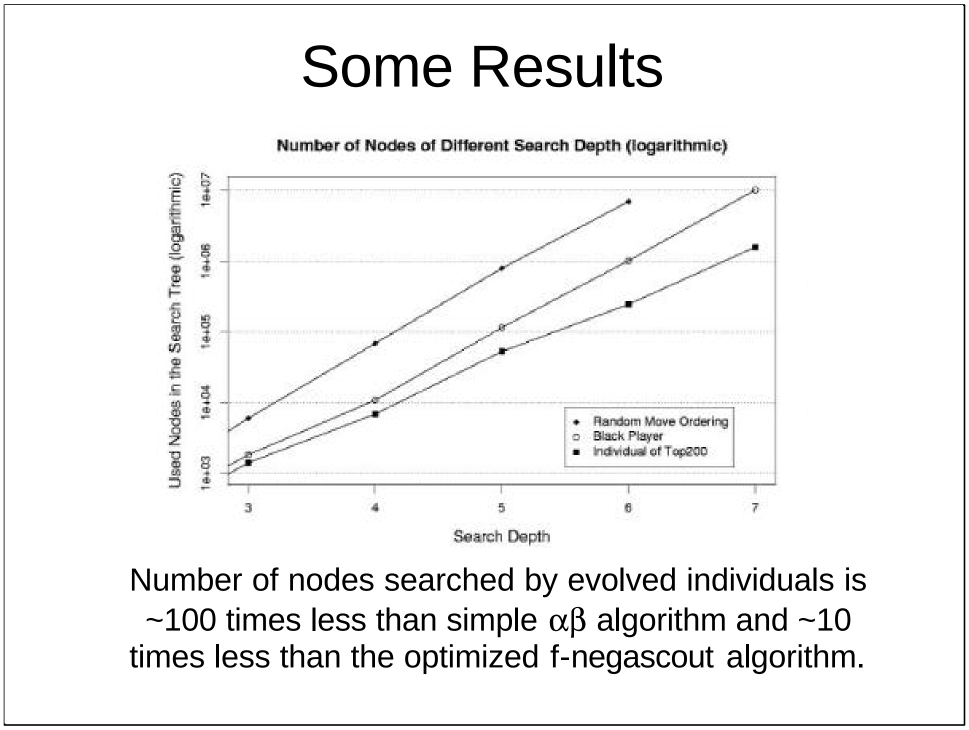
This ambitious move was successful because of the vast amount of resources available to the project. The chances of creating an even semi-viable depth module given only 100 population members would be vanishingly small, but when over 1,000,000 internet users downloaded EvoChess and created small populations of their own, the probability became appreciable. Indeed, the final evolved players at the end of the project had evolved a very complicated, intricate depth module that was efficient and powerful.



Evaluation Function Parameters

Much more complicated function than Kendall/Whitwell

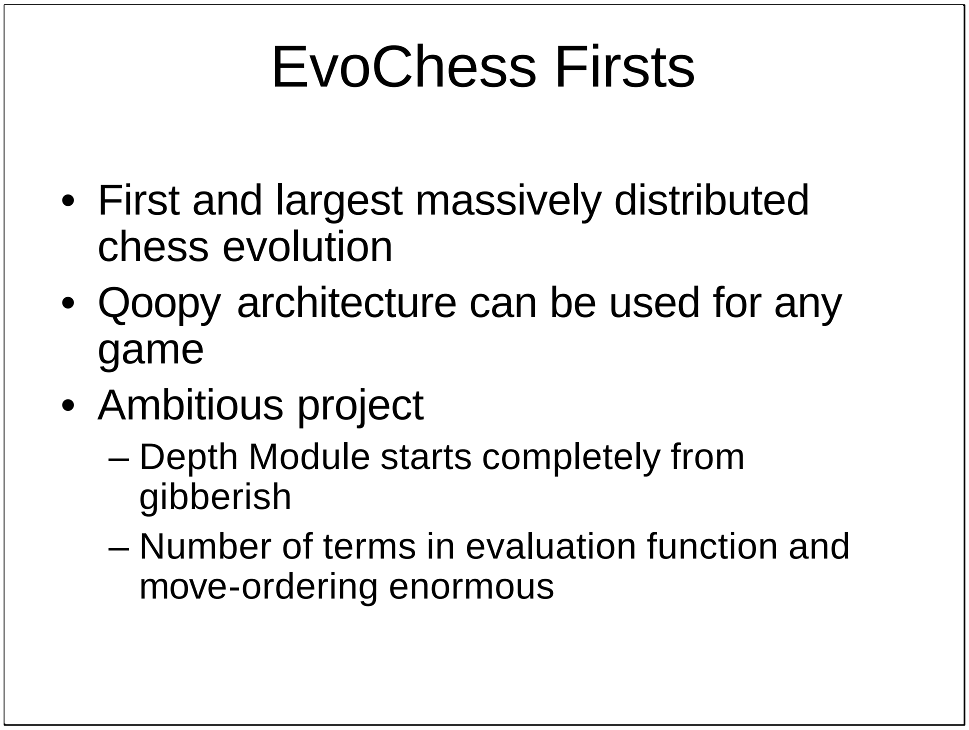
This slide shows the various parameters included in the EvoChess evaluation function. I show it for two reasons. First of all, given the complexity of this function and the rest of the chess-players it still astounds me that EvoChess was able to achieve a converged evolved solution of such prowess starting from initially random players. Secondly, for those who are interested, this is a fairly comprehensive list of the most important evaluation function parameters for creating algorithms that play chess. The inclusion of certain parameters is often disputed of course as well as their relative weightings. It is notable that these disputes can easily be settled through the use of evolutionary algorithms.



Some Results

Number of nodes searched by evolved individuals is ~100 times less than simple αβ algorithm and ~10 times less than the optimized f-negascout algorithm.

Once the EvoChess population had significantly evolved, it yielded exciting results. The graph above shows that a simple alpha-beta algorithm searches about 100 times more nodes than the EvoChess evolved individual, and even the state-of-the-art f-negascout algorithm searched 10 times more nodes. And yet the EvoChess individual can consistently defeat the f-negascout algorithm playing at a depth of 5-ply. (Since the EvoChess algorithm was limited to only 100,000 nodes, it would be expected to lose to an f-negascout that was allowed to search more than 1,000,000 nodes or more.



EvoChess Firsts

- First and largest massively distributed chess evolution
- Qoopy architecture can be used for any game
- Ambitious project
  - – Depth Module starts completely from gibberish
  - – Number of terms in evaluation function and move-ordering enormous

This slide recaps some of the major points I highlighted about EvoChess and why it was an innovative, significant project. First of all, it was the first internet-distributed chess evolution program of its kind. Its ambitious nature makes it a difficult stunt to top. A second benefit of this project was the development of the qoopy architecture. With only minor recoding, this architecture can easily be used to model the internet-evolution of other problems beside chess. Along with other board games that the authors of EvoChess have proposed, I would put forth the possibility of evolving things like future Mars mission parameters through such a distributed framework. If each user could help evolve a mission trajectory or mission architecture, perhaps an interesting optimum not yet thought of by humans could be reached. Finally, I re-emphasize that EvoChess was a very ambititous evolution project. Starting the depth modules from completely random gibberish was a daring move that aptly demonstrated the awesome power of genetic programs and must serve to silence many skeptics.

Sources

- Section 1: Alpha Beta Mini-Max:
  -

- Section 2: Inductive Adversary Modeler:
  - S. Walczak (1992) Pattern-Based Tactical Planning. International Journal of Pattern Recognition and Artificial Intelligence 6 (5), 955-988.
  - S. Walczak (2003) Knowledge-Based Search in Competitive Domains IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 15, NO. 3, 734 – 743

- Section 3: Evolutionary Algorithms
  - Kojima, et. al. *An Evolutionary Algorithm Extended by Ecological Analogy to the Game of Go.* Proceedings 15 Intl. Joint Conf. on AI, 1997.
  - Koza. *Genetic~~ Programming~~*~~. Encyclopedia of Com~~puter Science and Technology. 1997.
  - ~~Zbigniew. E~~*volutionary Algorithms for Engineering Applications*. 1997.