4. Game playing

4.1 Search problems

- *State space search* — find a (shortest) path from the initial state to the goal state.
- *Constraint satisfaction* — find a value assignment to a set of variables so that given constraints are met.
- *Combinatorial optimization* — find a value assignment to a set of variables so that an objective function is minimized (or maximized).
- *Games* find an optimal strategy to win.

4.2 Games and AI

- Good game-playing programs have been AI's triumphs.
- Game research interested in developing high-performance, real-time solutions.
- “Ends justify the means” -approach
- The illusion of human-like behavior result of
  - Brute-first search
  - Large memory
  - Discover knowledge from self-play
  - Simulation of scenarios
  - Comprehension without understanding
- Chess has been the fruit-fly of artificial intelligence.
- Special issue in games and puzzles: *Artificial Intelligence,* Volume 134, Issues 1-2 (2002)

4.3 Games of interest

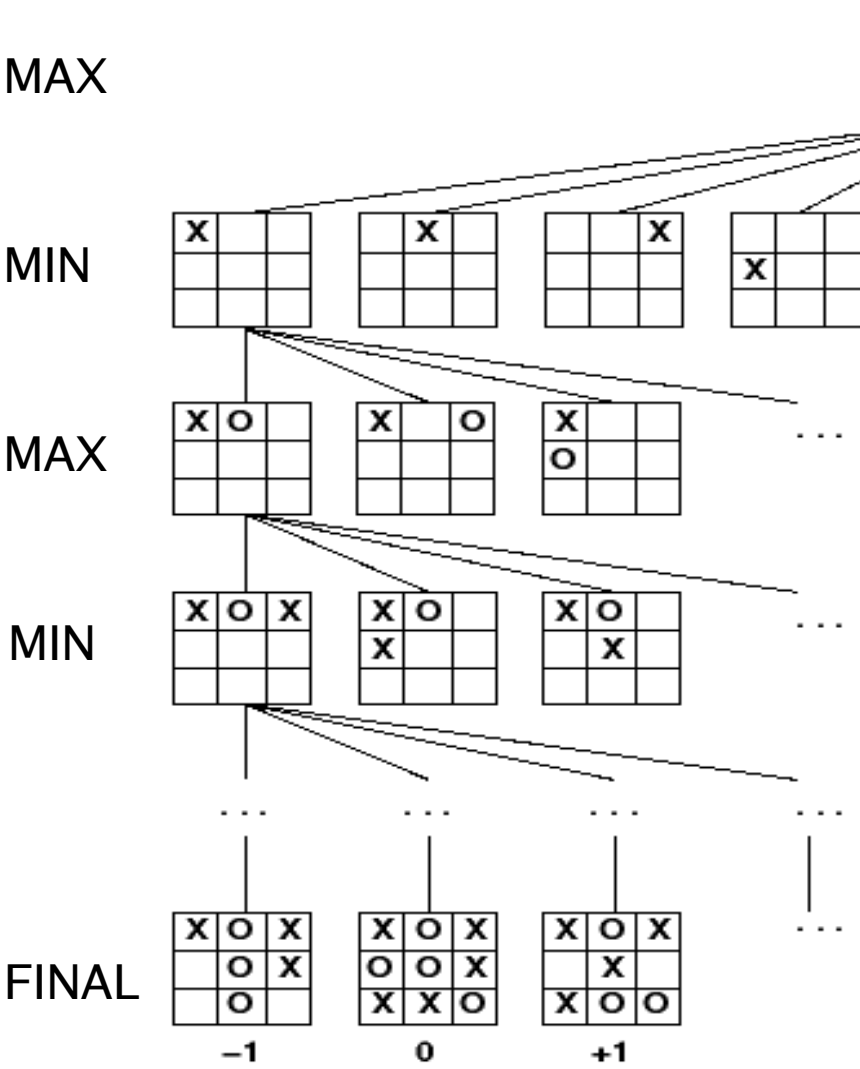
- Two-person zero-sum games; if one player wins, the other must lose.
- Players are adversarial — both not only try to win but cause the opponent to lose.
- No chance involved
- Complete information about
  - Available actions
  - Payoffs
- Examples: Tic-tac-toe, checkers, chess

4.4 Game tree

- Many two-player games can be represented as a tree called *game tree*.
  - Root of the tree represents the state when no moves have been made yet (*starting position*), with the indicator who moves first.
  - Nodes represent possible states of the game (positions).
  - Arcs represent legal moves in the game.
  - Leaf nodes represent the final states, when the game has been won, lost or ended up in draw.
- Moves by two players are represented as alternate levels in the tree*; ply* refers to the depth of the tree.
- *Utility function* gives a numeric value to the outcome of the game.
- *Evaluation function* enables the player to estimate if a given state is good or bad.

4.5 Example tree

TIC-TAC-TOE

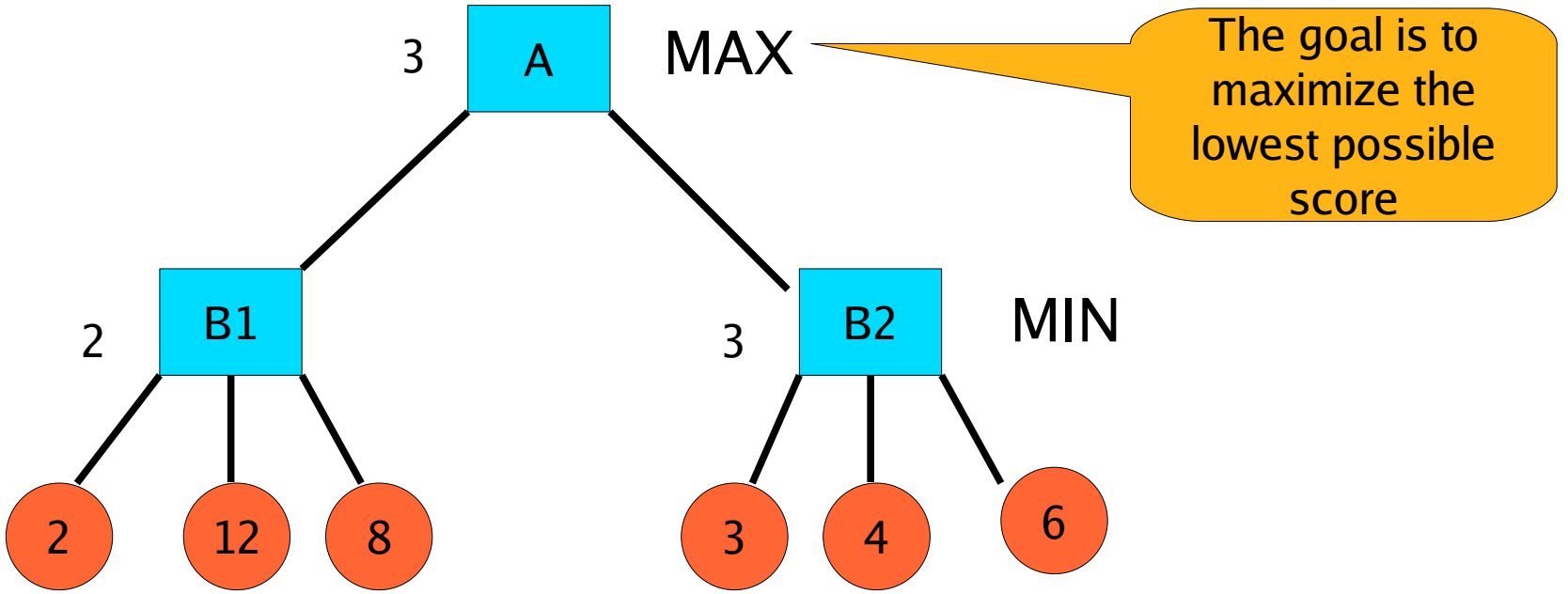


Players are:

1. player = MAX → maximize one's own payoff
2. player = MIN → minimize other's (MAX's) payoff

4.6 Minimax strategy

- Proceeds depth-first
- Assumes the opponent plays rationally, i.e., makes the best moves for her.
- Determines the optimal strategy for MAX:
  - Generate the whole game tree
  - Propagate utility values from leaves toward the root.
- The whole game tree needs to be examined.



4.7 Searching game trees

- One cannot use exhaustive search
  - The tree is potentially huge.
  - The opponent complicates the search.
- One can use depth-first or breadth-first search to generate the tree, but other methods need to be used to choose good moves:
  - Bounded lookahead
  - Alpha-beta pruning

4.8 Bounded look-ahead

- At each move the search tree is examined to particular depth (cut-off point).
- Nodes at cut-off are considered leaf nodes and evaluated using a static evaluation function.
- Difficulty of choosing a fixed cut-point:
  - Non-quiescent positions in near future $\rightarrow$ cut search only at points that are safe, i.e., unlikely to cause a large change in relative positions of two players in near future.
  - Horizon problem; consequences of a bad move is postponed beyond the search depth $\rightarrow$ no general solution exists.

4.9 Alpha-beta pruning

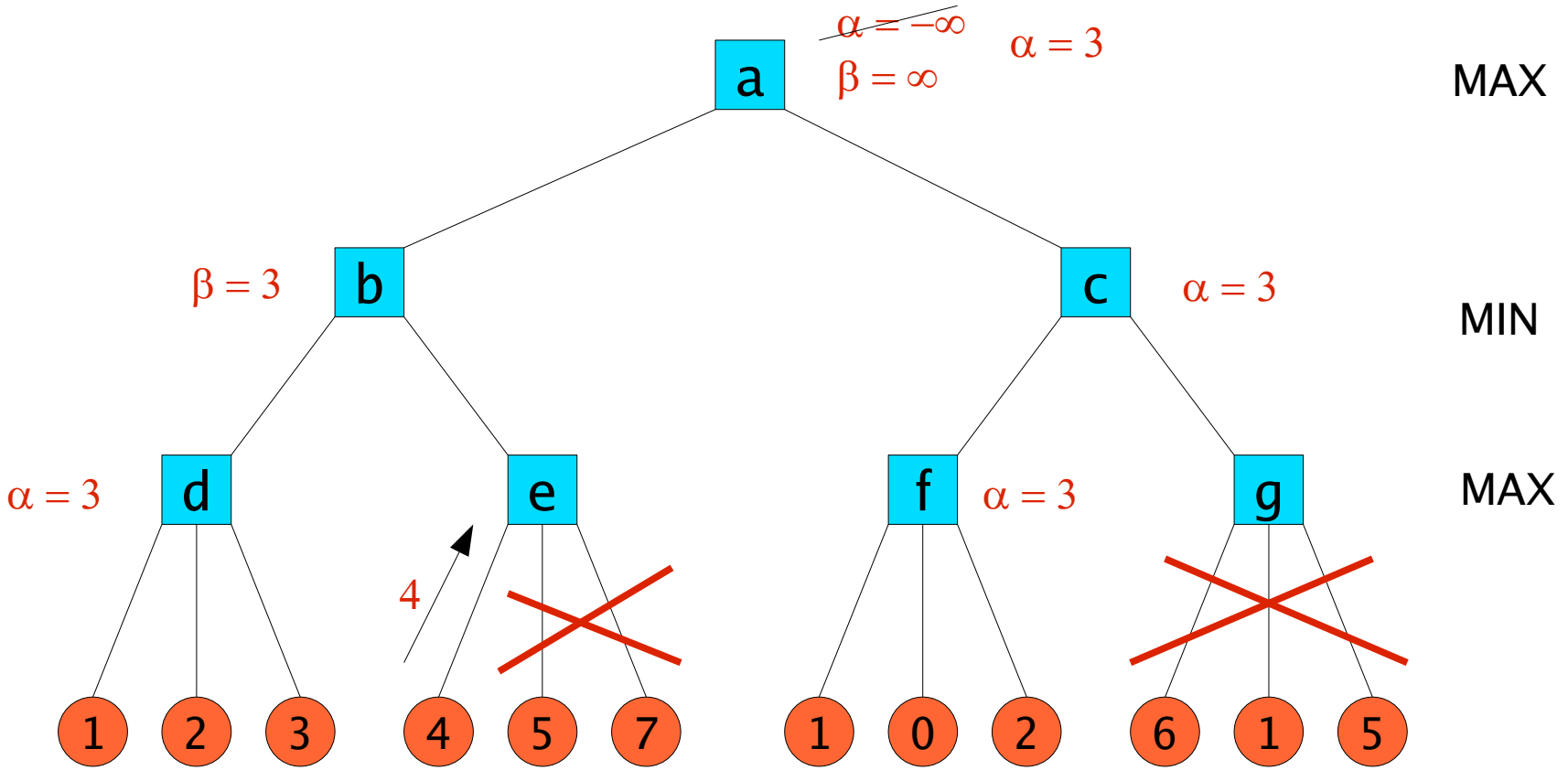
- Remove sections of the game tree that are not worth examining.
- In other words, if a move is determined to be worse than another move already been examined, it is pointless to examine the consequences of the worse move.
- Does not change the outcome of the game, if both players play optimally.
- Effectiveness depends on the order the nodes are evaluated; best performance when the best choice at each level is always the first one.
- For MAX node
  - α = maximum value found in its descendants
  - β = minimum beta value found in its MIN ancestors
- For MIN node
  - β = minimum value found in its descendants
  - α = maximum alpha value found in its MAX ancestors

4.10 Alpha-beta algorithm

- Traverses the game tree in depth-first order.
- Function alpha_beta(current_node, *alpha*, *beta*)

```
If ROOT(current_node)
    alpha = -inf
    beta  = inf
If LEAF(current_node)
    return payoff
If MAX_node
    alpha = max(alpha, alpha_beta(children, alpha, beta))
    If alpha ≥ beta
        cut_off(current_node)
If MIN_node
    beta = min(beta, alpha_beta(children, alpha, beta))
    If beta ≤ alpha
        cut_off(current_node)
```

4.11 Alpha-beta example

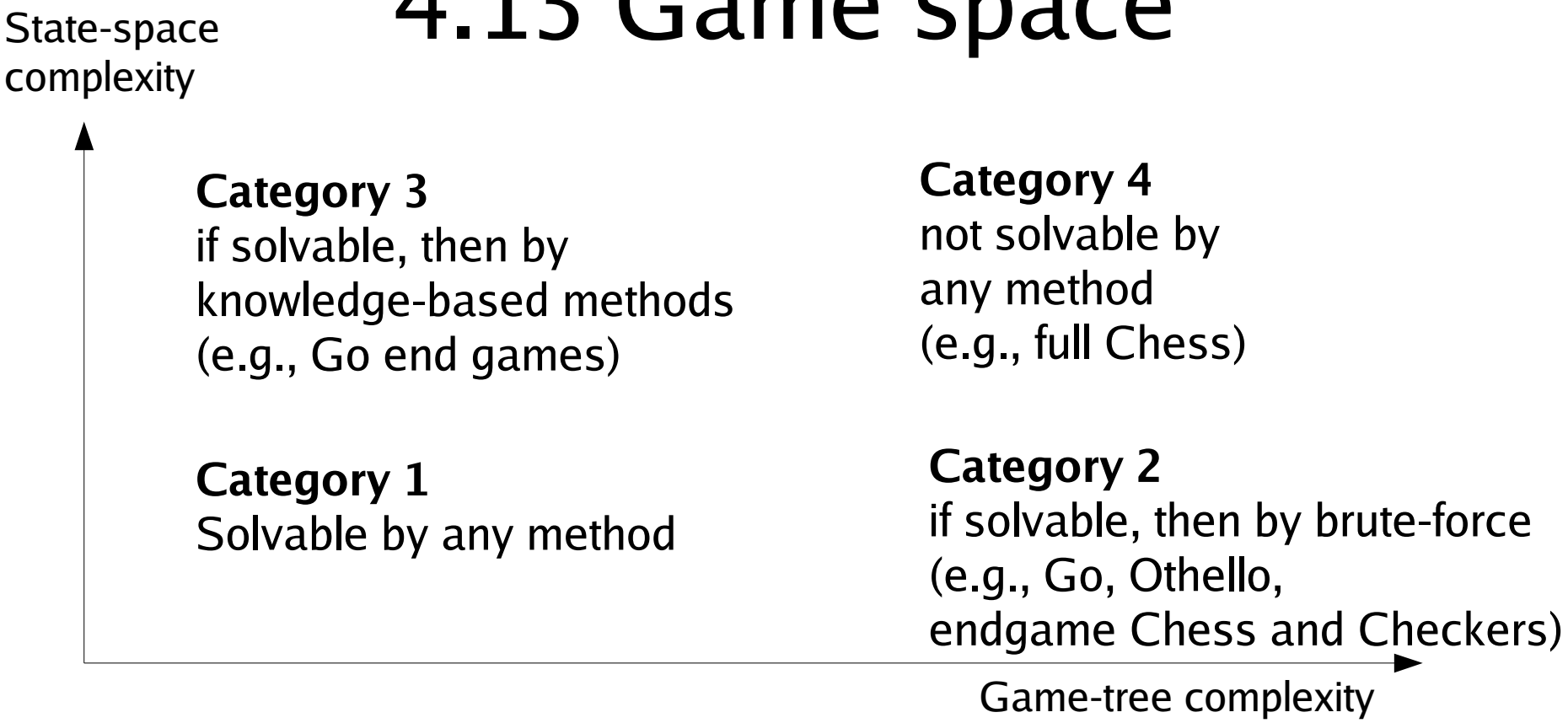


From MAX player's point of view

4.12 Solving games

- Finding the *game-theoretic value* of the game (van den Herik *et al.*, 2002)
  - Determining the final result of the game when neither player makes a mistake, i.e., they play perfectly.
  - Value indicates if the first mover wins, loses or the game ends in draw.
- Three levels:
  - *Ultra-weekly solved*: the perfect-play result is known, but not the strategy
  - *Weekly solved*: both the result and the strategy are known from the beginning of the game.
  - *Strongly solved*: the result can be computed for all possible positions in the game.
- Why? To explore if the knowledge from solving games can be translated to rules and strategies that
  - can be applied by humans.
  - are general and not ad hoc.
  - are transferable between games.

4.13 Game space



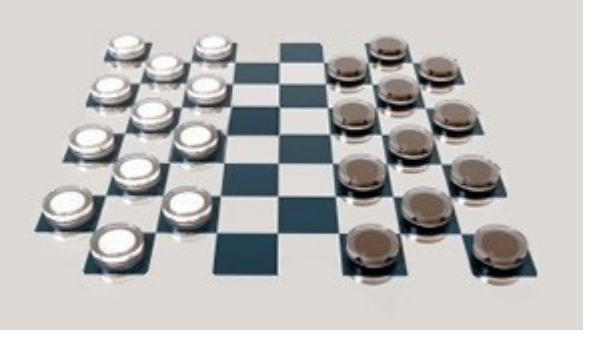
- State space complexity = number of legal positions reachable from initial position
- Game-tree complexity = number of leaf nodes in the solution search tree of the initial position.
- Convergent (Chess, Checkers) vs. divergent games (Go, Othello)

4.14 Case study I: Chinook

- Checkers-playing program by Jonathan Schaeffer (Univ. of Alberta, 1989)
- Played on 8 x 8 board, on dark squares only with two types of pieces → average branching factor = 8
- Checkers (weakly) solved (2007): the game ends in draw if both players play perfectly → the program never loses.
- Games against world champions:
  - 1990 lost
  - 1992 won 2, lost 4, 33 drawn
  - 1994 all six games drawn
  - 1995 → won

Marion Tinsley, the world champion for 40 years

Don Lafferty, world's 2nd best



4.15 Chinook's strategy

- Examines game tree to depth of 20, about 1000 positions per second.
- Uses Minimax with alpha-beta pruning, iterative deepening, and applies a number of heuristics.
- Database of opening moves
  - Better strategy: use a anti-book containing moves that should be avoided, and search for good moves $\rightarrow$ surprise factor and imaginative play
- Database of endgame positions
  - Evaluation of all possible positions with 8 pieces (~440 billion)
- Linear weighted static evaluation function:
  - Piece count, king count, distribution of pieces on the board, etc.
  - Game divided into 4 phases depending on the number of pieces: different weights in each phase.

- Study of Chinook revealed a serious shortcoming of minimax with alpha-beta pruning.
  - Most possible games lead to draws
  - Not all draws are equal:
    - a position may lead to a draw no matter moves the player makes.
    - another position may lead to draw if player makes a correct choice at each move.
- A new version of minimax take advantage of the difficulty of certain paths through game.

4.16 Case study II: Logistello

- Othello-playing program by Michael Buro (PhD thesis, 1994)
- Beat human champion Takeshi Murakami in 1997. Now retired.
- Techniques to improve move quality in limited-time tournaments:
  - Very high search speed by special purpose hardware
  - Smart evaluation function
  - Selective search
  - Large opening book
- Incompatibilities and trade-offs between the above factors; still can improve existing implementation by attacking them separately.

- Position evaluation
  - Approximation of perfect evaluation function = combinations of a few distinct numerical properties of a position –features .
  - Selection is the most important and hardest task. Requires:
    - Domain-specific knowledge
    - Programming skills
  - Generalized Linear Evaluation Model (GLEM)
    - ~100 000 binary features (e.g., "is a white disc on h8?")
    - 1.2 million automatically fitted parameters

- Selective search
  - Values returned by minimax searches to different depths highly correlated
  - Uses the value found by shallow search to estimate the minimax value from deeper search.
  - If the estimate is worse than can be expected from alpha-beta search there is no need to search deeper.
- Opening book learning
  - Save time, avoid falling to known strategic traps, prevents from losing twice in the same way.
  - Copy the opponent's winning moves next time.

4.17 Availability of information

- *Complete information*: every player knows the payoffs and the strategies available to other players (type of players, structure of the game)
- *Perfect information*: player knows the actions of the other players (what happens within the game)
- *Certain information*: players know which game they are playing, i.e., what the payoff from a certain strategy will be given the strategies played by others.
- Games of incomplete and imperfect information pose problems to search based methods.
  - Games of chance

4.18 Other kind of games

- Two-person perfect-information games (e.g., Chess, Othello)
- Multiple-player, stochastic, incomplete or imperfect information games (e.g., Poker, Backgammon)
- Interactive games, such as action games, role-playing games, adventure games, and sports games are a topics of a whole another course!

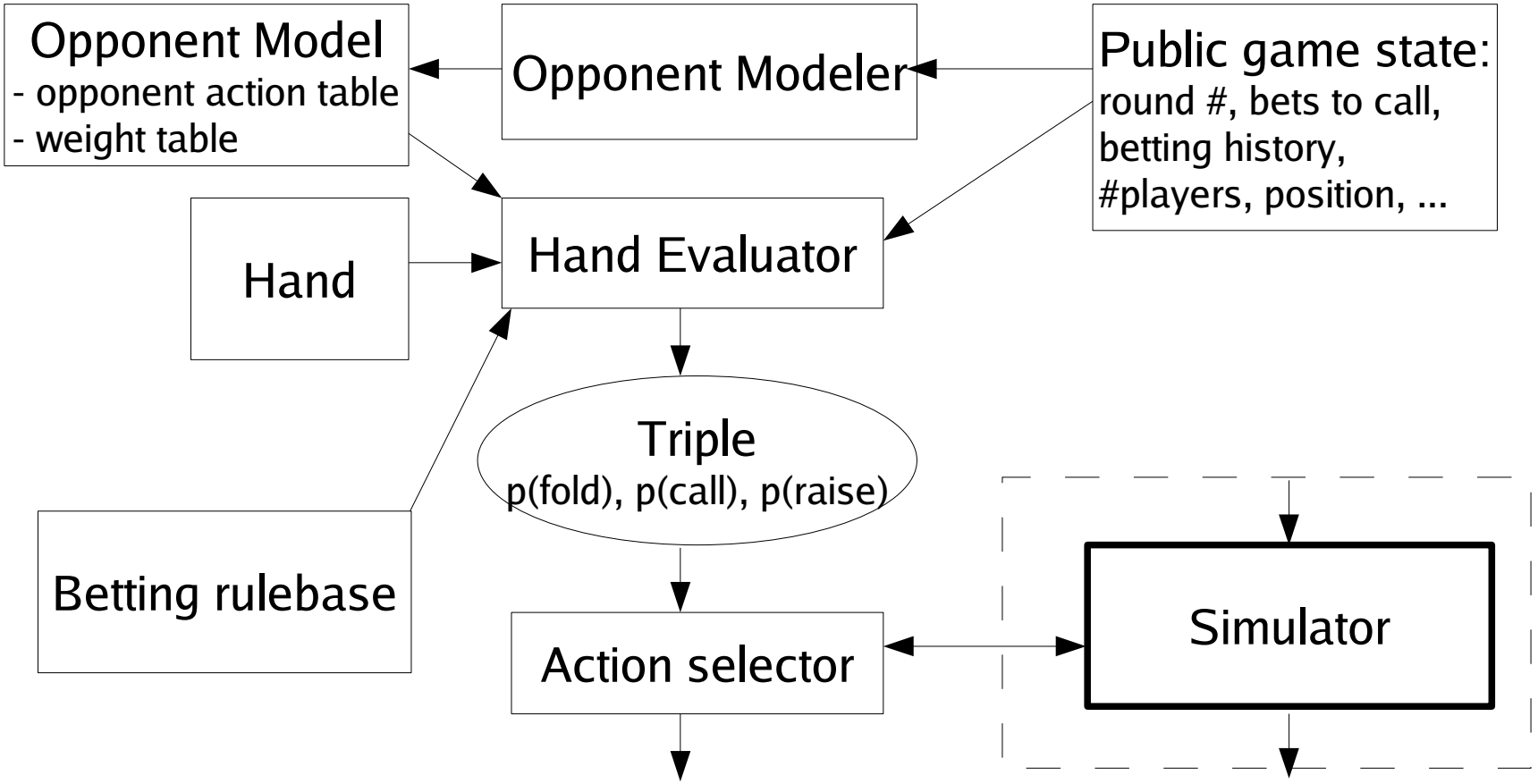
4.19 How to study these games?

- Simplified versions
  - Subset of the game, e.g., in poker two players, 3 cards, one betting round, two betting decisions.
  - Address each sub-problem separately
  - Abstractions; collect similar sub-problems into same class
- Tackle whole problem at once (Billings *et al.*, 2002).
- For instance, for poker this includes
  - Betting strategy
  - Opponent modeling
  - Learning
  - Performance evaluation

4.20 Case study III: Poki

- Plays Texas Hold'em (Billings *et al.*, 1999, 2002)
- Game of imperfect information
  - Risk assessment
  - Deception
- Multiple players, dealer
- Tries to attack the whole problem at once.

4.21 Poki architecture



4.22 Poki: Learning and decision making

- Basic betting strategy:
  1. Compute effective hand strength=current hand strength+potential to improve
  2. Calculate probabilities of actions: fold, call, and raise
  3. Choose action stochastically
- Simulation based betting strategy; play out many likely scenarios to get the expected value of each betting action.
- Opponent modeling
  - Deduce the strength of the hand from actions
  - Predict future actions
    - General Opponent Model (GOM): Fixed strategy based on rational choice
    - Specific Opponent Model (SOM): Personal history

4.23 Case study IV: TD-Gammon

- Blackgammon-playing program by Gerald Tesauro (1995)
- Learns, using temporal difference learning, by playing against itself.
- Discussed in more detail next week.