Definitions and Constraints

- two players
  - first player A(lice)
  - second player B(ob)
- turn based
  - A, B, A, B
- both players have complete information
- no randomness
- positions (states)
  - finite set of positions with one or more starting positions
  - repeating moves (infinite loops) are considered draws
- moves (transition from one position to the next)
  - each position has a set of possible moves/next positions
    - potentially no legal move
  - normal play
    - first player who cannot move loses
  - every game ends after a finite number of moves
    - e.g. chess prevents the same move 3 times in a row
    - some exceptions exist
- might be asymmetric
  - e.g. Fuchs und Henne
  - [[Examples of Combinatorial Games]]

First-Player and Second-Player Win games

- some games favor the first (starting) or second player
- one player may have a major advantage due to the starting position or being able to move first or second
- therefore this player always wins
  - assuming both players play optimally

Levels of Game Solutions

- ultra-weakly solved
  - known who wins but not how
- weakly solved
  - strategy is known
  - must be followed from the very start on
- strongly solved

  - known from any valid state
- ultra-strongly solved
  - know for any move during any game state whether it wins/loses/draws
  - also know in how many half-moves

Game-Tree vs State-Space Complexity

- **Game-Tree Complexity** Number of nodes the complete decision tree for a whole game has

  State-Space Complexity Number of states which can be reached from the start state by valid moves

game	state-space complexity	game-tree complexity	branching factor
Tic Tac Toe	10^3	10^5	5
Nine Men's Morris	10^{10}	10^{50}	10-30
Pyraos	10^{11}	10^{33}	9
Awari	10^{12}	10^{32}	5-6
Connect-4	10^{14}	10^{21}	5-7
Abalone	10^{25}	10^{180}	65-70
Reversi	10^{28}	10^{58}	5-15
Chess	10^{50}	10^{123}	35
Go	10^{171}	10^{360}	300-400

Storing Game States

- needs to be efficient and complete
- move generator
  - creates successors of game states
- identify final states
  - win
  - lose
  - draw
- equivalent game states
  - allow transitions to the same successor state
  - must not be perfectly identical
    * reflections
    * rotations
    * inversion
    * color-change
  - fingerprint/canonical state

* store only one of the equivalent states
* ???

Processing Game States

Initialize S with the starting state

$\forall$ non-processed states $s \in S$ DO
/* process newly added states */
    $\forall$ successors t of s DO
        compute canonical state t' of t
        IF $t' \notin S$ THEN add t' to S

/* S contains all states which are reachable from the start state via valid moves */

- state code
  - non-negative integer

    WIN: code odd: number of half-moves in which a win can be forced (if player plays perfect).

    LOSE: code even: number of half-moves in which the game is at most lost (if opponent plays perfect).

    DRAW: special code, e.g. -1; no number of half-moves possible.

  - draw does not contain the number of half moves
    * due to circles/infinite loops
    * exceptions exist such as Connect 4
  - determine action based on code

    Init all states without valid moves (with code 0, draw, ...)
    /* terminal states without successors */

    IF successor state with even code exists THEN
        code := (smallest even code of a succesor state) + 1
        /* WIN in that number of moves */
    ELSE IF successor state with draw code exists THEN
        code := draw
        /* DRAW */
    ELSE
        code := (largest (odd) code of a successor state) + 1
        /* LOSE in that number of moves */

  - compute codes

```
Init all states without valid moves
/* terminal states without successors */
Init all remaining states with 'undefined'
                                        max-depth ... until no new codes can be de
FOR k := 1 TO max-depth /* k = # of half-moves */
  ∀ states s ∈ S with still undefined code DO
    IF k is odd THEN
      IF s has a successor with code k − 1 THEN
        code of s is k /* WIN state */
    ELSE /* k is even */
      IF all successors of s have odd codes THEN
        code of s is k /* LOSE state */
Set all 'undefined' states to draw.
```

*