Game Theory

Motivation

- examine the role of AI methods in games
- some game provide challenges that can be formulated as abstract competitions with clearly defined states and rules
  - programs for some games can be derived from search methods
  - narrow view of games
- games can be used to demonstrate the power of computer-based techniques and methods
- more challenging games require the incorporation of specific knowledge and information
- expansion of the use of games
  - from entertainment to training and education

Objectives

- explore the combination of AI and games
- understand the use and application of search methods to game programs
  - apply refined search methods such as minimax to simple game configurations
  - use alpha-beta pruning to improve the efficiency of game programs
  - understand the influence of chance on the solvability of chance-based games
- evaluation of methods
  - suitability of game techniques for specific games
  - suitability of AI methods for games

Games and Computers

games offer concrete or abstract competitions

- "I'm better than you!"

some games are amenable to computer treatment

- mostly mental activities
- well-formulated rules and operators
- accessible state

others are not

- emphasis on physical activities
- rules and operators open to interpretation
  - need for referees, mitigation procedures
- state not (easily or fully) accessible

Defining Computer Games

video game / computer game

"a mental contest, played with a computer according to certain rules for amusement, recreation, or winning a stake" [Zyda, 2005]

serious game

a mental contest, played with a computer in accordance with specific rules, that uses entertainment to further training or education [modified from Zyda, 2005]

[Zyda, 2005] Michael Zyda, "From Visual Simulation to Virtual Reality to Games," *IEEE Computer, vol. 39, no. 9, pp. 25-32.*

Aspects of Computer Games

- story
  - defines the context and content of the game
- art
  - presentation of the game to the user
    - emphasis on visual display, sound
- software
  - implementation of the game on a computer
- purpose
  - entertainment
  - training
  - education

Games and AI

- traditionally, the emphasis has been on a narrow view of games
  - formal treatment, often as an expansion of search algorithms
- more recently, AI techniques have become more important in computer games
  - computer-controlled characters (agents)
  - more sophisticated story lines
  - more complex environments
  - better overall user experience
- Games (especially chess) have been a great focus for AI research.Why?
  - Uncertainty makes them like the real world
  - The rules can be well defined
  - The game position is easily represented
  - There are no moral or ethical problems
  - We can match computers against both other computers and humans
  - Playing games is seen as an "intelligent activity

Cognitive Game Design

- story development
  - generation of interesting and appealing story lines
  - variations in story lines
  - analysis of large-scale game play
- character development
  - modeling and simulation of computer-controlled agents
  - possibly enhancement of user-controlled agents
- immersion
  - strong engagement of the player's mind
- emotion
  - integration of plausible and believable motion in characters
  - consideration of the user's emotion
- pedagogy
  - achievement of "higher" goals through entertainment

Game Analysis

- often deterministic
  - the outcome of actions is known
  - sometimes an element of chance is part of the game
    - e.g. dice
- two-player, turn-taking
  - one move for each player
- zero-sum utility function
  - what one player wins, the other must lose
- often perfect information
  - fully observable, everything is known to both players about the state of the environment (game)
  - not for all games
    - e.g. card games with “private” or “hidden” cards
    - Scrabble

Games as Adversarial Search

- many games can be formulated as search problems
- the zero-sum utility function leads to an adversarial situation
  - in order for one agent to win, the other necessarily has to lose
- factors complicating the search task
  - potentially huge search spaces
  - elements of chance
  - multi-person games, teams
  - time limits
  - imprecise rules

Difficulties with Games

- games can be very hard search problems
  - yet reasonably easy to formalize
  - finding the *optimal* solution may be impractical
    - a solution that beats the opponent is "good enough"
  - unforgiving
    - a solution that is "not good enough" not only leads to higher costs, but to a loss to the opponent
- example: chess
  - size of the search space
    - branching factor around 35
    - about 50 moves per player
    - about 35^{100} or 10^{154} nodes
      - about 10^{40} *distinct* nodes (size of the search graph)

Search Problem Formulation

- initial state
  - board, positions of pieces
  - whose turn is it
- successor function (operators)
  - list of *(move, state)*
  - defines the legal moves, and the resulting states
- terminal test
  - also called goal test
  - determines when the game is over
  - calculate the result
    - usually win, lose, draw, sometimes a score (see below)

utility or payoff function

- numeric value for the outcome of a game

Single-Person Game

- conventional search problem
  - identify a sequence of moves that leads to a winning state
  - examples: Solitaire, dragons and dungeons, Rubik's cube
  - little attention in AI
- some games can be quite challenging
  - some versions of solitaire
  - a heuristic for Rubik's cube was found by the Absolver program

Contingency Problem

- uncertainty due to the moves and motivations of the opponent
  - tries to make the game as difficult as possible for the player
    - attempts to maximize its own, and thus minimize the player's utility function value
  - different from contingency due to neutral factors, such as
    - chance
    - outside influence

Two-Person Game

- How do we think when we play chess?
- We consider making a move....
  - *if I move my queen there, then my opponents' best move is to move their knight there, etc. etc.*
- We are making some assumptions:
  - we want to make our best possible move
  - our opponent is as skillful as we are
  - our opponent has the same information as us
  - our opponent will also do their best to win

Two-Person Game

- games with two opposing players
  - often called MIN and MAX
  - usually MAX moves first, then they take turns
  - in game terminology, a *move* comprises one step, or *ply* by each player
- MAX wants a strategy to find a winning state
  - no matter what MIN does
- MIN does the same
  - or at least tries to prevent MAX from winning
- full information
  - both players know the full state of the environment
- partial information
  - one player only knows part of the environment
  - some aspects may be hidden from the opponent, or from both players

Perfect Decision in Games

- Consider two players of a game, MAX and MIN
  - MAX moves first
- The game begins in an *initial state*
- There is a set of *operators*, i.e. legal moves
- *Terminal states*, where the game ends
- each terminal state has a *utility function*, i.e. a pay-off to each player
- The **Utility** is assumed to be in relation to MAX
  - we assume the Utility function(Evaluation function) is symmetrical, i.e. what is good for MAX is equally bad for MIN and vica-versa
- Examples of utility
  - chess might have a utility of 1 for a win, 0 for a draw and -1 for a loss
  - Zero sum games

Simple Games

- Assume that we can generate and search the entire decision tree
  - this is only possible for simple games, later we'll think about more complex games
- How should MAX move?
  - some lines end in wins for MAX, some in wins for MIN, but we don't know how MIN will move...

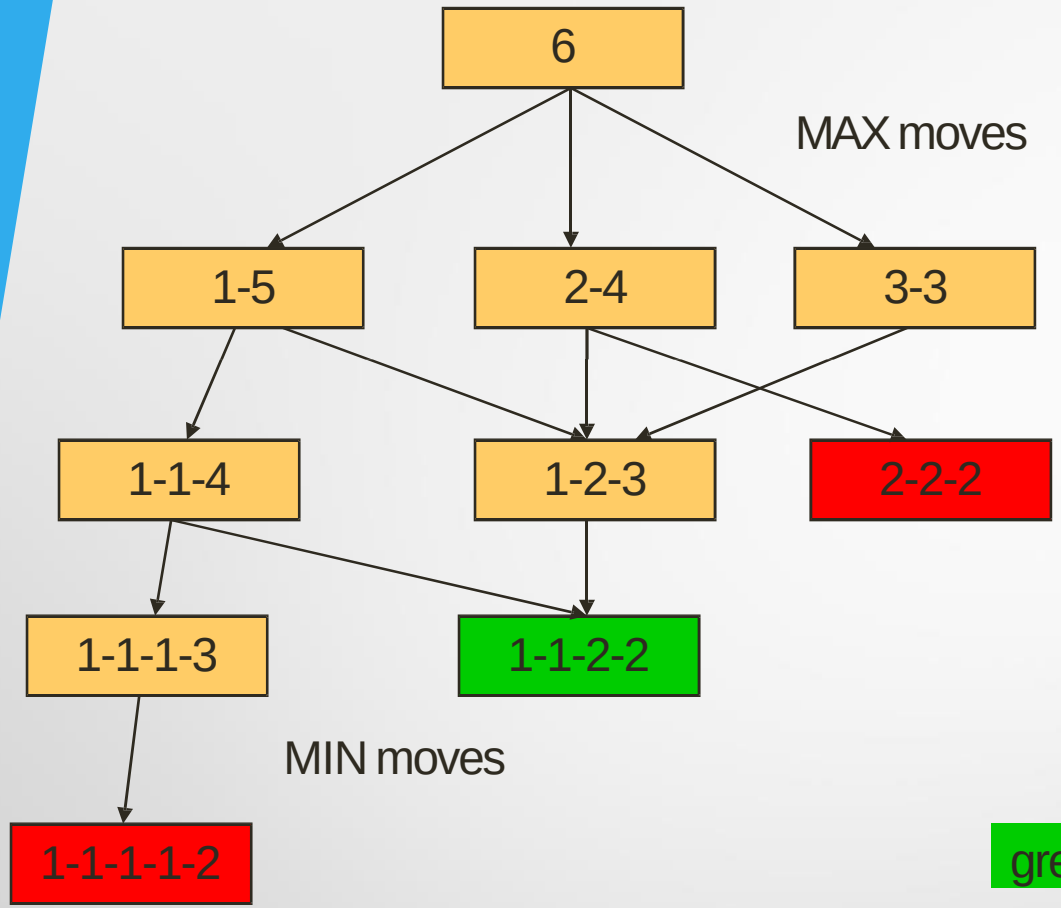
MINIMAX Searching

- What we do is first build the game tree
- Then we work in a bottom-up fashion, starting down at the terminal states
- What we assume is as follows:
  - When MAX is moving, MAX will chose the line with the highest utility, so pass up the MAXimum utility to the next highest level
  - When MIN is moving, MIN will chose the line with the lowest utility (for MAX), so pass up the MINimum utility to the next highest level
- When we reach the current position, MAX choses the line of highest utitlity (as usual) and moves

Example of MinIMax

- A simple game is the game of ***nim***
- The rules of *nim* are as follows:
  - we start with a number of sticks in a group
  - on each move, a player must divide a group into two smaller groups
  - groups of one or two sticks can't be divided
  - the last player who makes a legal move wins

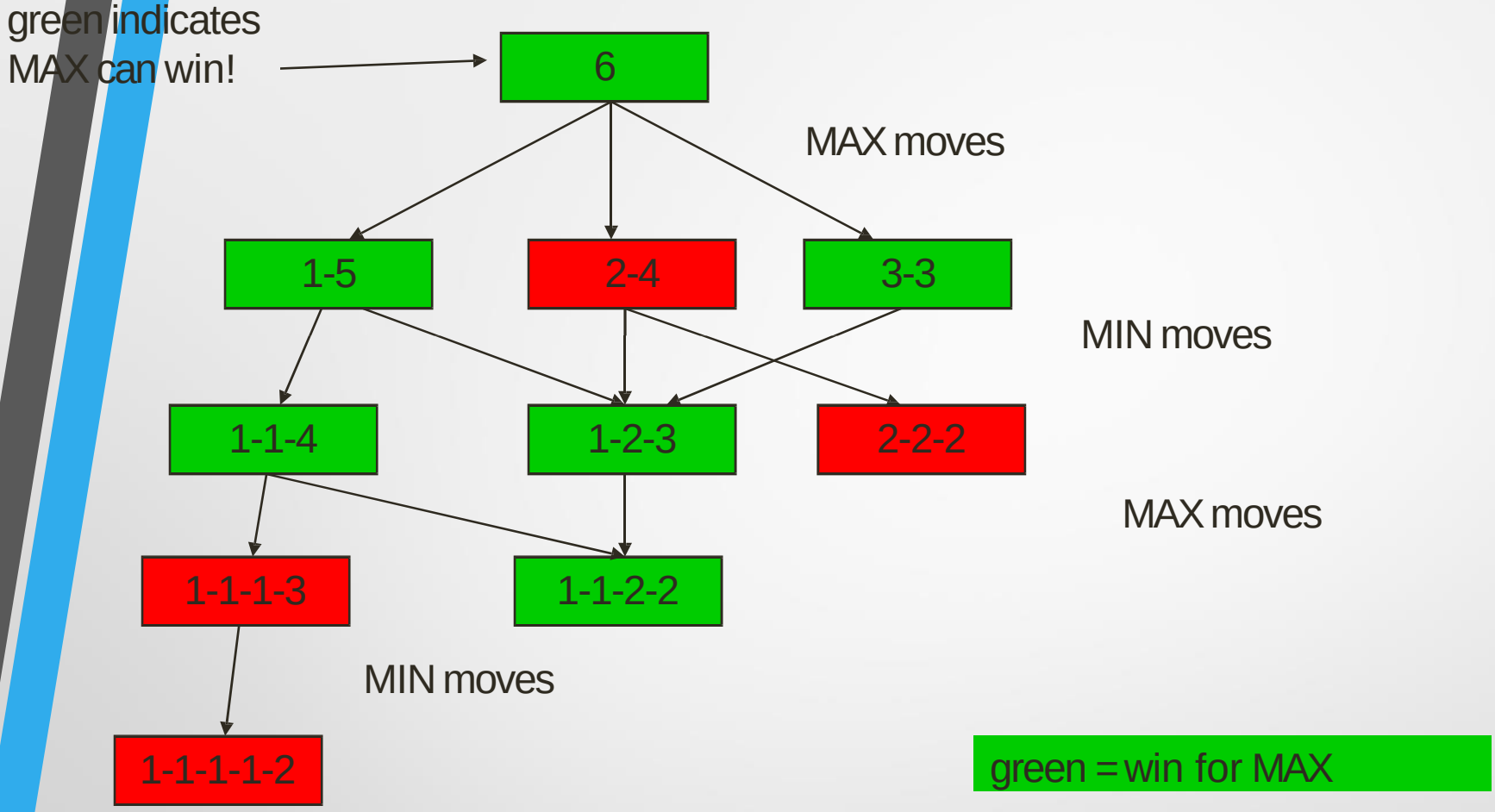
NIM Search Tree



green = win for MAX

red = win for MIN

MINIMAX Search of NIM



green = win for MAX

red = win for MIN

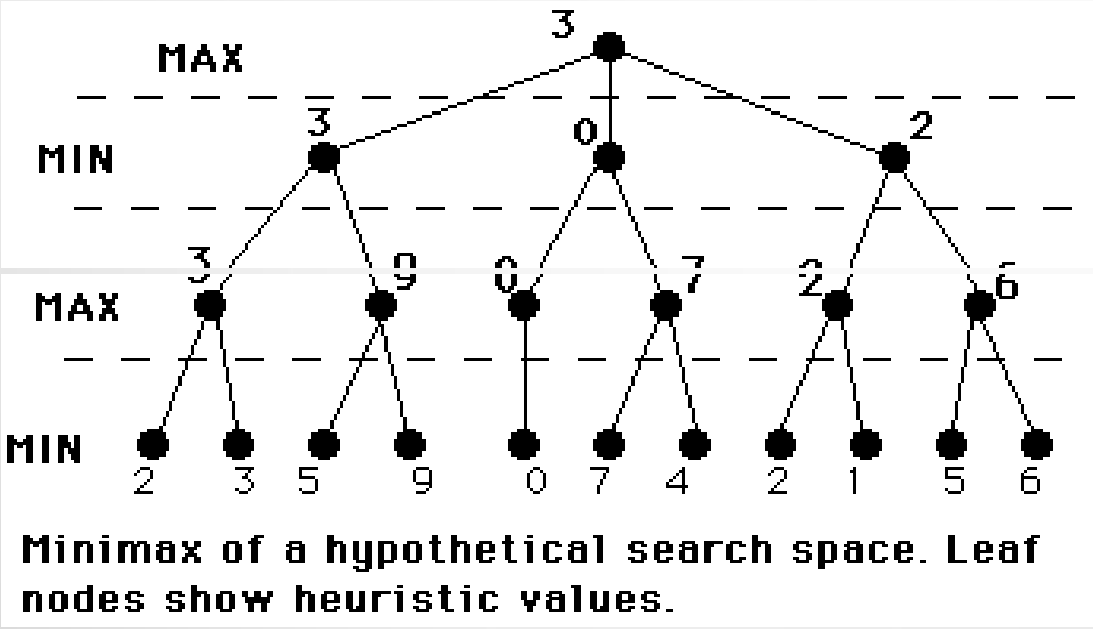
Heuristic MINIMAX

- If the search tree is too large, we can't go down to the terminal states
- In this case we can search down only a certain number of levels (this is called the *PLY* of the search)
- At the maximum depth,we use a heuristic *evaluation function* to rate the utility of the different positions
  - then use the MINIMAX as usual to decide what to do

Pratical Aspects of heuristic MINIMAX

- When we take each step, we will only need to calculate the evaluation function for the next level, provided we have stored the tree
  - this will save time but costs space
- Usually we search the tree depth-first
- We need to trade off sophistication of heuristics *vs* cost of searching

Heuristic MINIMAX



Minimax of a hypothetical search space. Leaf nodes show heuristic values.

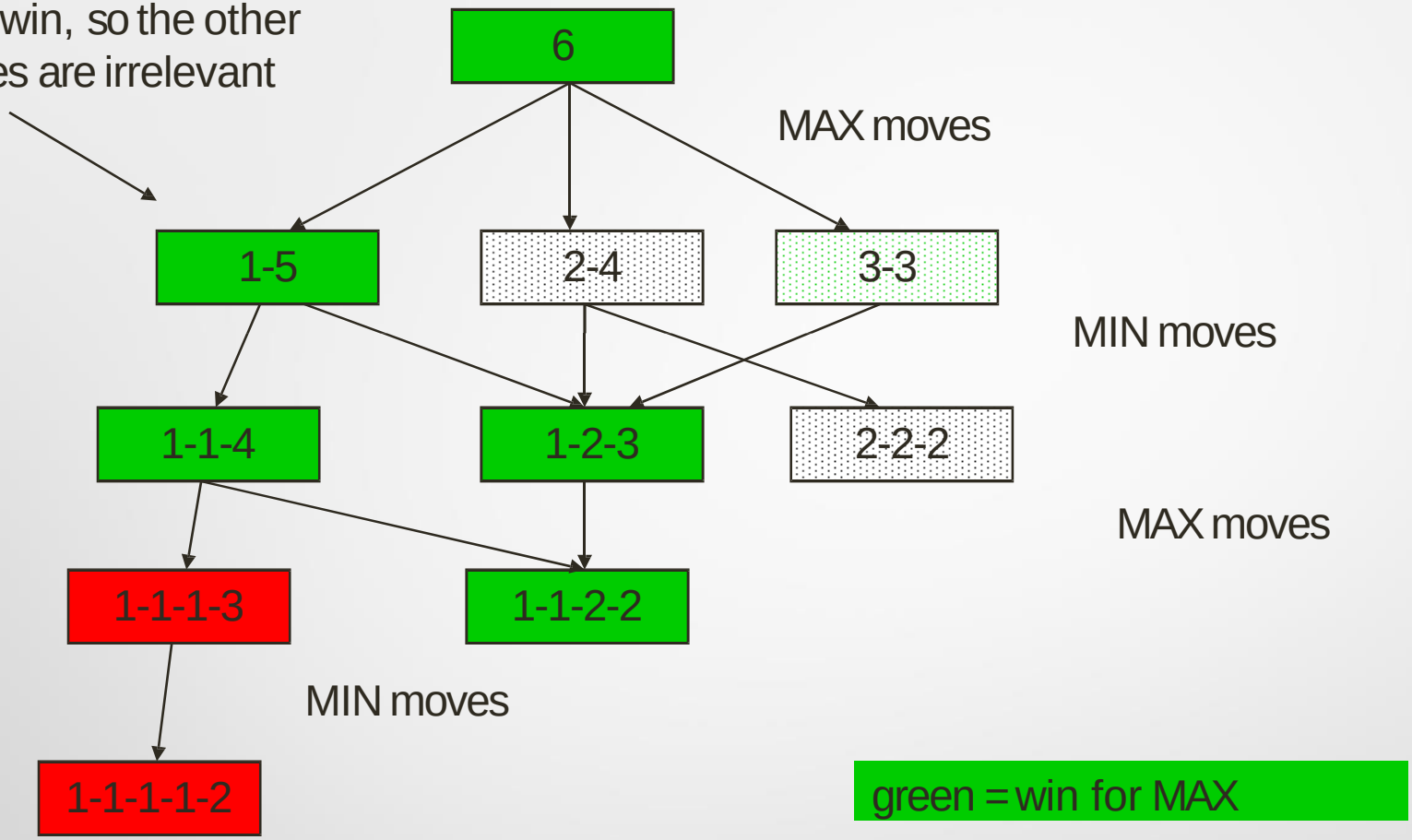
PRUNING

- When carrying out MINIMAX, we can save a lot of work without loss of performance by *pruning*

- Remember we are searching the tree depth first:
  - so in an instance where we have a simple discrete utility function, as soon as we have found a way to force a win, we don't need to look any longer
  - this applies at all levels when MAX is to move - as soon as we find a guaranteed winning line, we can prune other options
  - similarly, when MIN moves, we can prune if we find a guaranteed losing line (i.e.a win for MIN)

Pruning the NIM Search

We have found a certain win, so the other branches are irrelevant

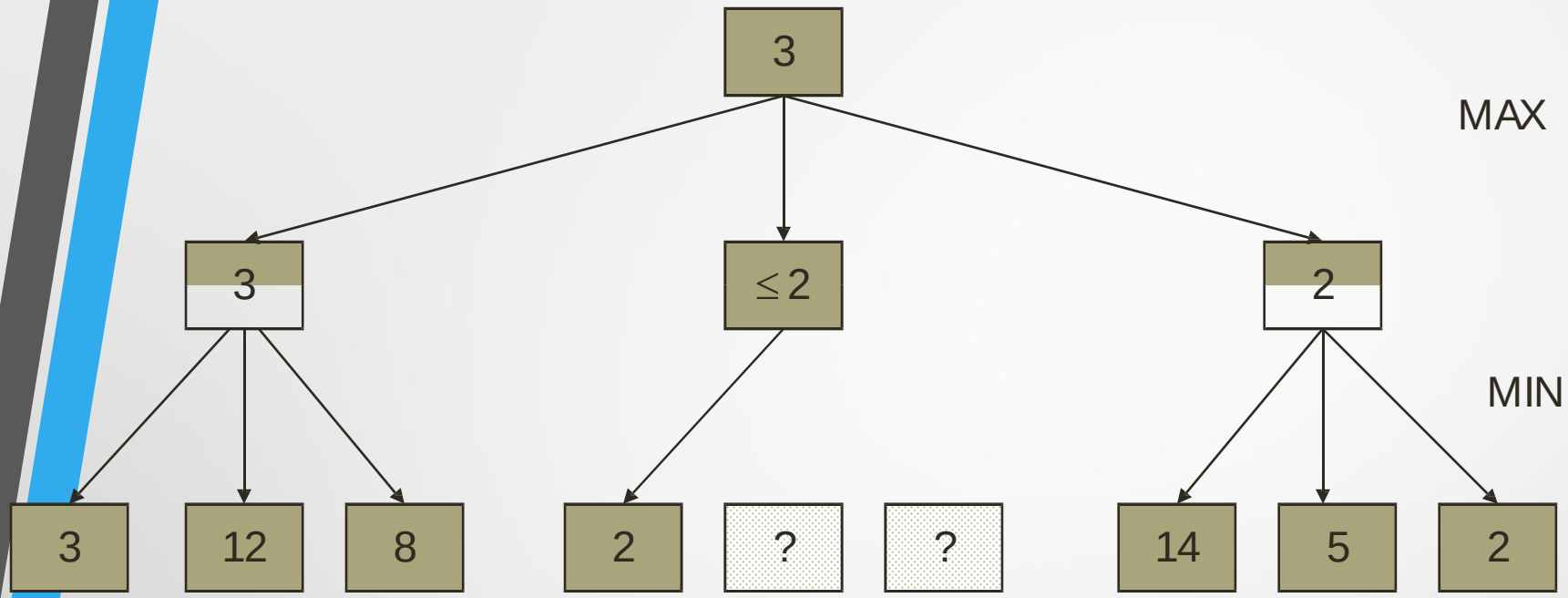


green = win for MAX

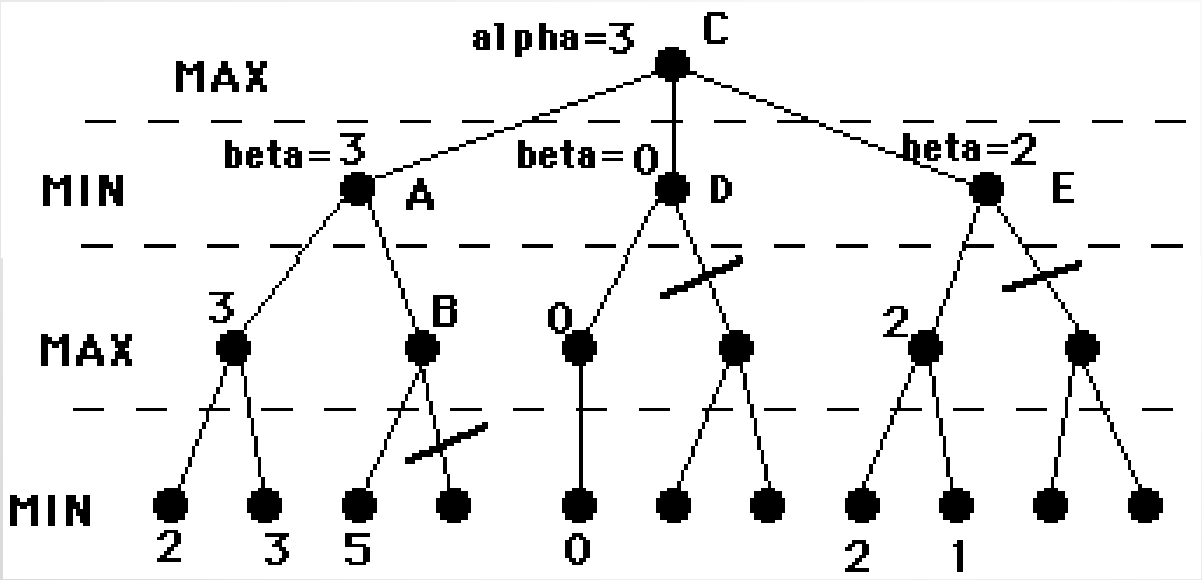
red = win for MIN

Alpha-Beta Pruning

- We can generalise pruning for *continuous* utility functions through a procedure known as α-β
- During the depth first search point we remember:
  - the score for the best choice so far for MAX = α
  - the score for the best choice so far for MIN = β
- When evaluating moves for MAX, if we reach a point in the tree where MIN can choose a move with utility $\leq \alpha$, we can prune the move above
  - MAX will not choose this move but prefer the chain to where α was found (unless a better branch exists)
  - similarly for MIN


3
MAX
3
≤ 2
2
MIN
3
12
8
2
?
?
14
5
2

Alpha Beta Pruning



Alpha Beta Pruning

1. Start at C. Descend to full-ply depth and assign the heuristic to a state and all siblings (MIN 2, 3). Back up these values to their parent node (MAX 3).

2. Offer this value to the grandparent (A), as its beta value. So, **A has beta=3**. A will be no larger than 3

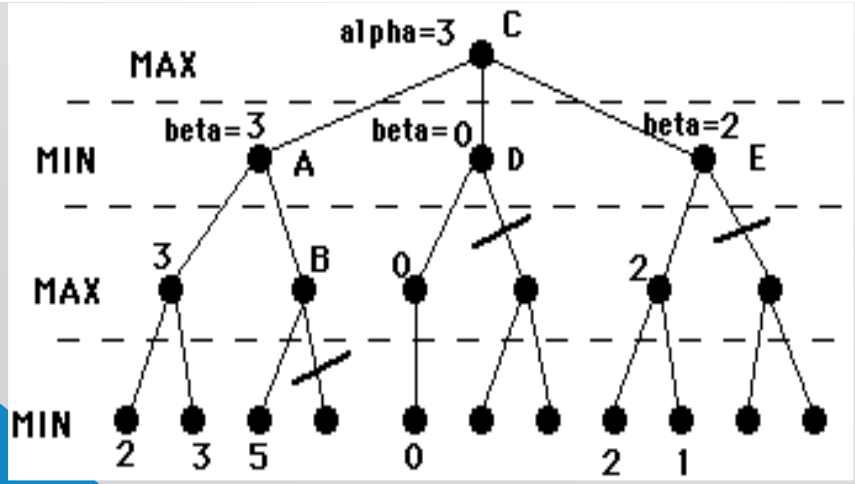
3. Descend to A's other grandchildren. Terminate the search of their parent if any grandchildren is >= A's beta. **Node B is beta-pruned**, as shown, because its value must be at least 5.

4. Once A's value is known, offer it to its parent (C) as its alpha value. So **C has alpha=3**. C will be no smaller than 3.

5. Repeat this process, descending to C's great grandchildren (0) in a depth-first fashion. **D is alpha-pruned**, because no matter what happens on its right branch, it cannot be greater than 0.

6. Repeating on E, **E is alpha-pruned** because its beta value (2) is less than its parent's alpha value (3). So no matter what happens on its right branch, E cannot have a value greater than 2.

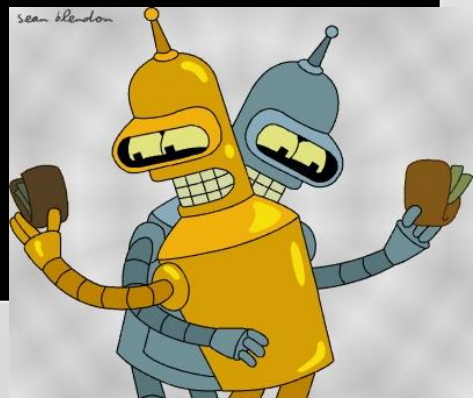
7. Therefore **C is 3**.



Alpha-Beta Algorithm

```
function Max-Value(state, alpha, beta) returns a utility value
  if Terminal-Test (state) then return Utility(state)
    for each s in  Successors(state) do
    alpha := Max (alpha, Min-Value(s, alpha, beta))
       if alpha  >= beta then return beta
    end
    return alpha

function Min-Value(state, alpha, beta) returns a utility value
  if Terminal-Test (state) then return Utility(state)
    for each s in  Successors(state) do
   beta := Min (beta, Max-Value(s, alpha, beta))
       if beta  <= alpha then return alpha
    end
    return beta
```



Properties of Alpha-Beta Pruning

- in the ideal case, the best successor node is examined first
  - results in $O(b^{d/2})$ nodes to be searched instead of $O(b^d)$
  - alpha-beta can look ahead twice as far as minimax
  - in practice, simple ordering functions are quite useful
- assumes an idealized tree model
  - uniform branching factor, path length
  - random distribution of leaf evaluation values
- transpositions tables can be used to store permutations
  - sequences of moves that lead to the same position

requires additional information for good players

- game-specific background knowledge
- empirical data

Imperfect Decisions

- complete search is impractical for most games
- alternative: search the tree only to a certain depth
  - requires a cutoff-test to determine where to stop
    - replaces the terminal test
    - the nodes at that level effectively become terminal leave nodes
  - uses a heuristics-based evaluation function to estimate the expected utility of the game from those leave nodes

Evaluation Function

- determines the performance of a game-playing program
- must be consistent with the utility function
  - values for terminal nodes (or at least their order) must be the same
- tradeoff between accuracy and time cost
  - without time limits, minimax could be used
- should reflect the actual chances of winning
- frequently weighted linear functions are used
  - $E = w_1 f_1 + w_2 f_2 + \ldots + w_n f_n$

    combination of features, weighted by their relevance

Example: Tic-Tac-Toe

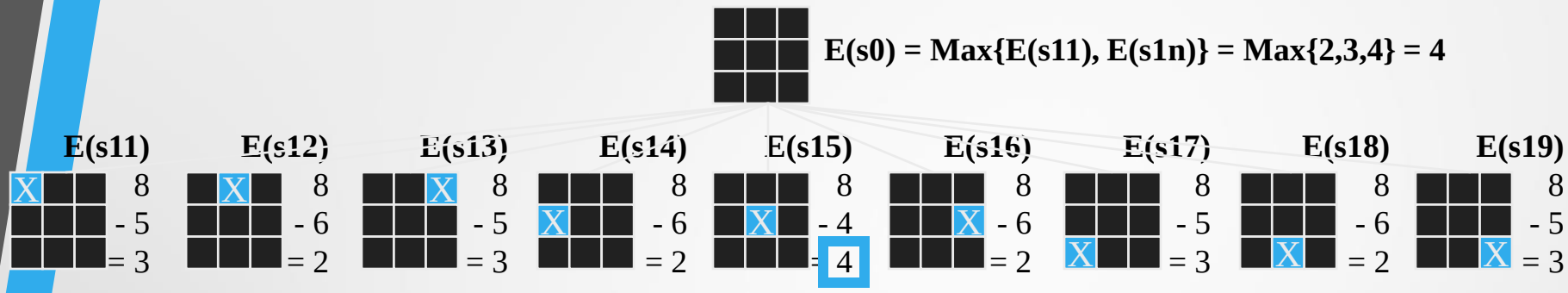
- simple evaluation function

```
E(s) = (rx + cx + dx) - (ro + co + do)
```

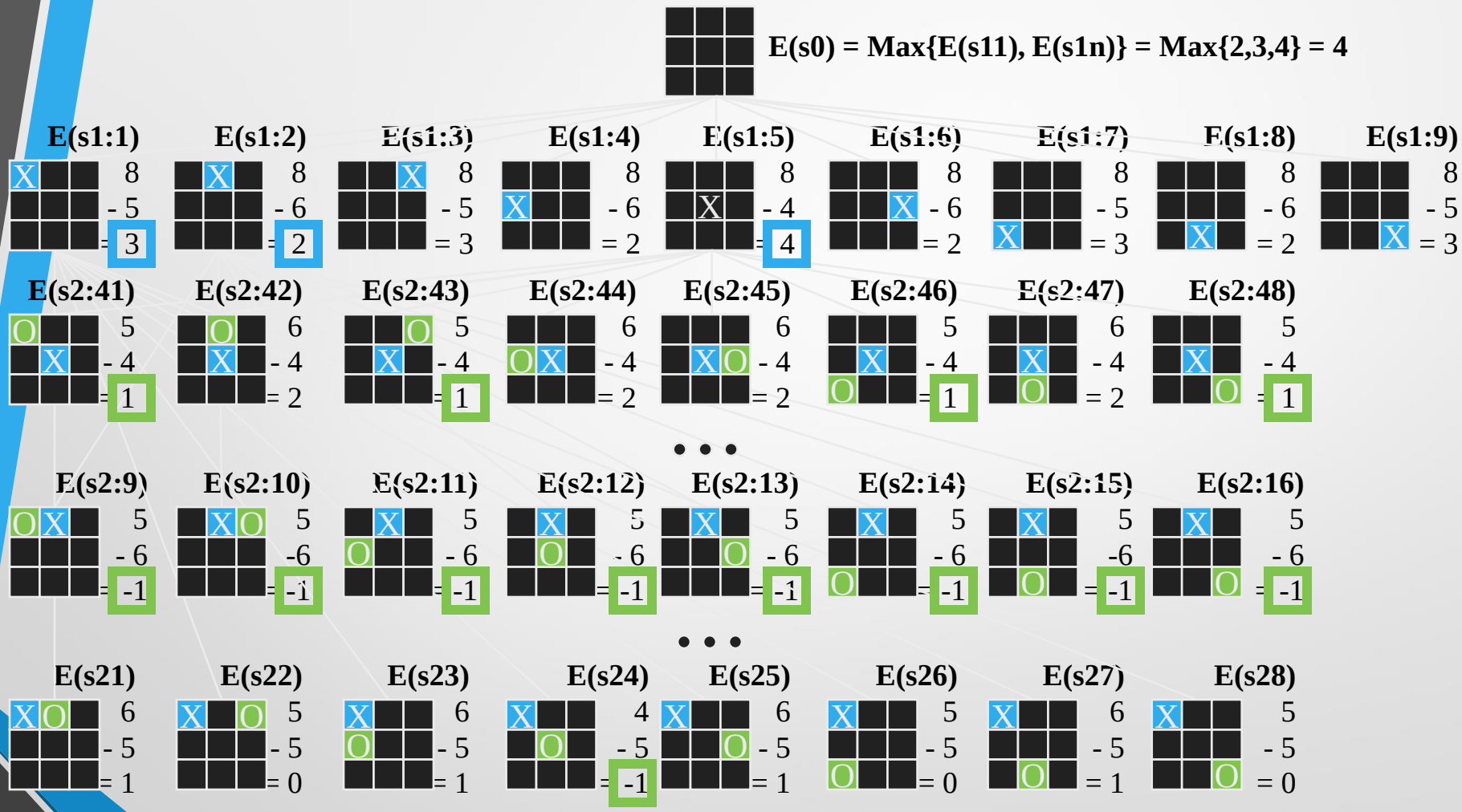
where r,c,d are the numbers of row, column and diagonal lines still available; x and o are the pieces of the two players

- 1-ply lookahead
  - start at the top of the tree
  - evaluate all 9 choices for player 1
  - pick the maximum E-value
- 2-ply lookahead
  - also looks at the opponents possible move
    - assuming that the opponents picks the minimum E-value

Tic-Tac-Toe 1-Ply



Tic-Tac-Toe 2-Ply



Checkers Case Study

- initial board configuration
  - Black single on 20
    single on 21
    king on 31
  - Red single on 23
    king on 22
  - evaluation function
    $E(s) = (5\,x_1 + x_2) - (5r_1 + r_2)$

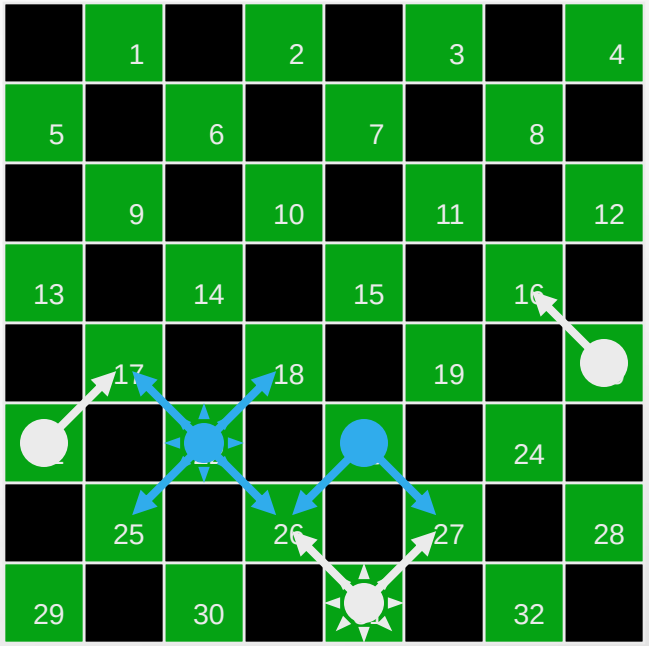
    where

    x_1 = black king advantage,

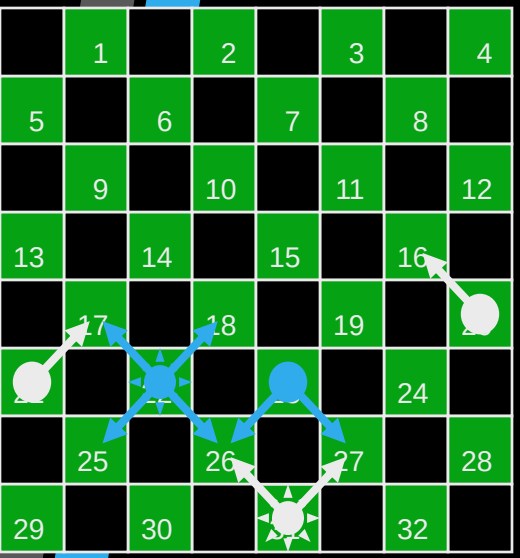
    x_2 = black single advantage,

    r_1 = red king advantage,

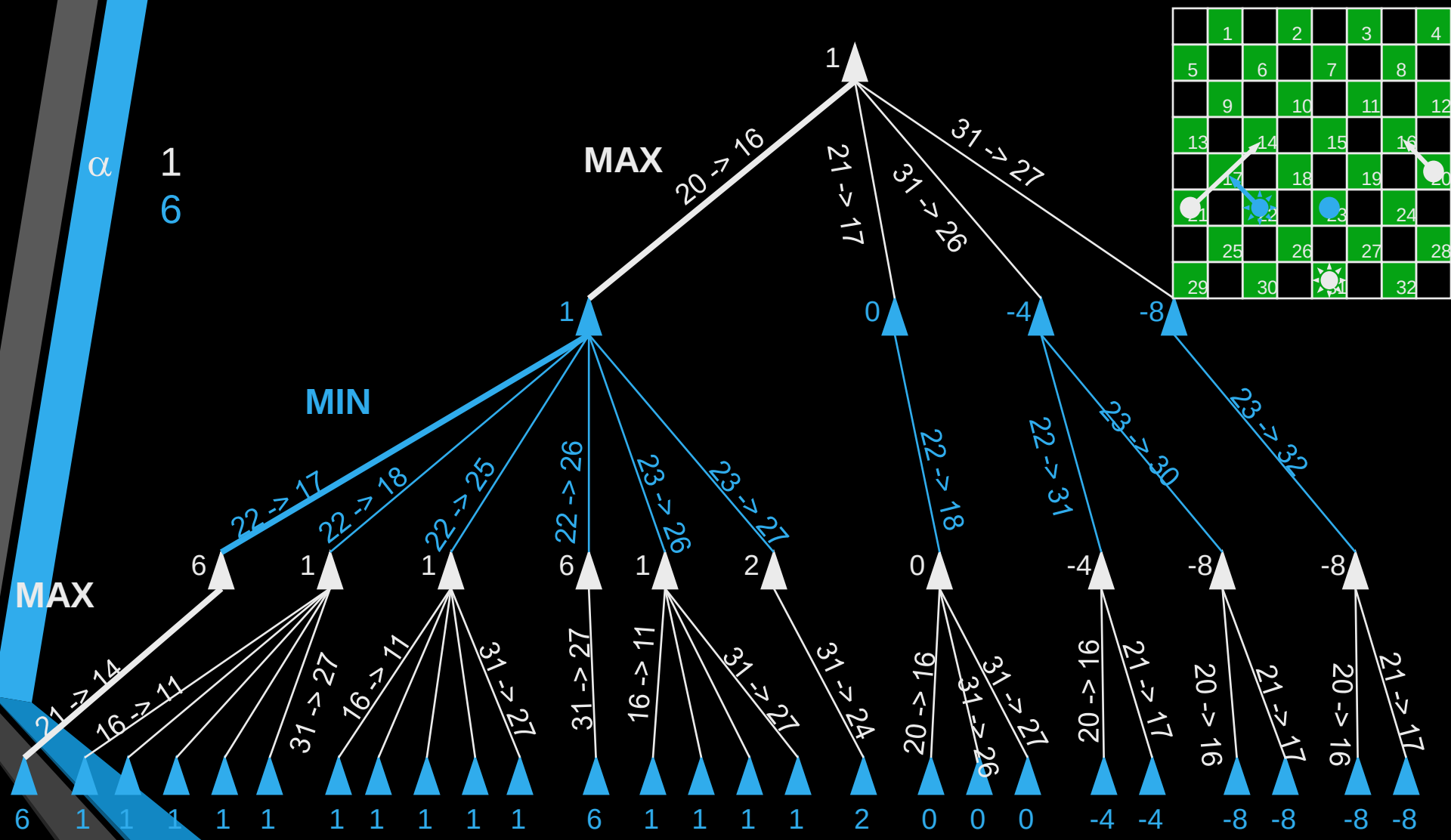
    r_2 = red single advantage



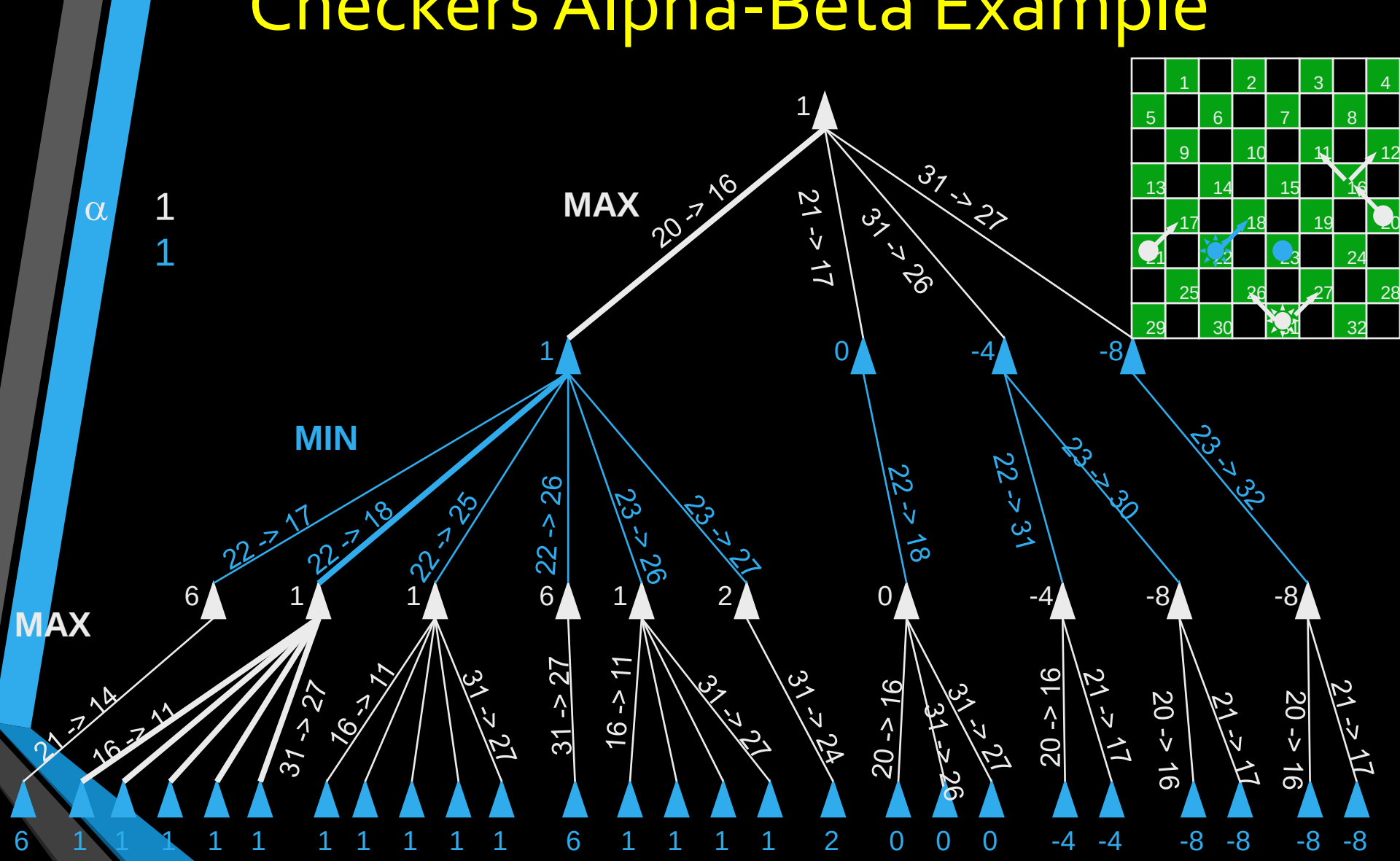
Checkers MiniMax Example



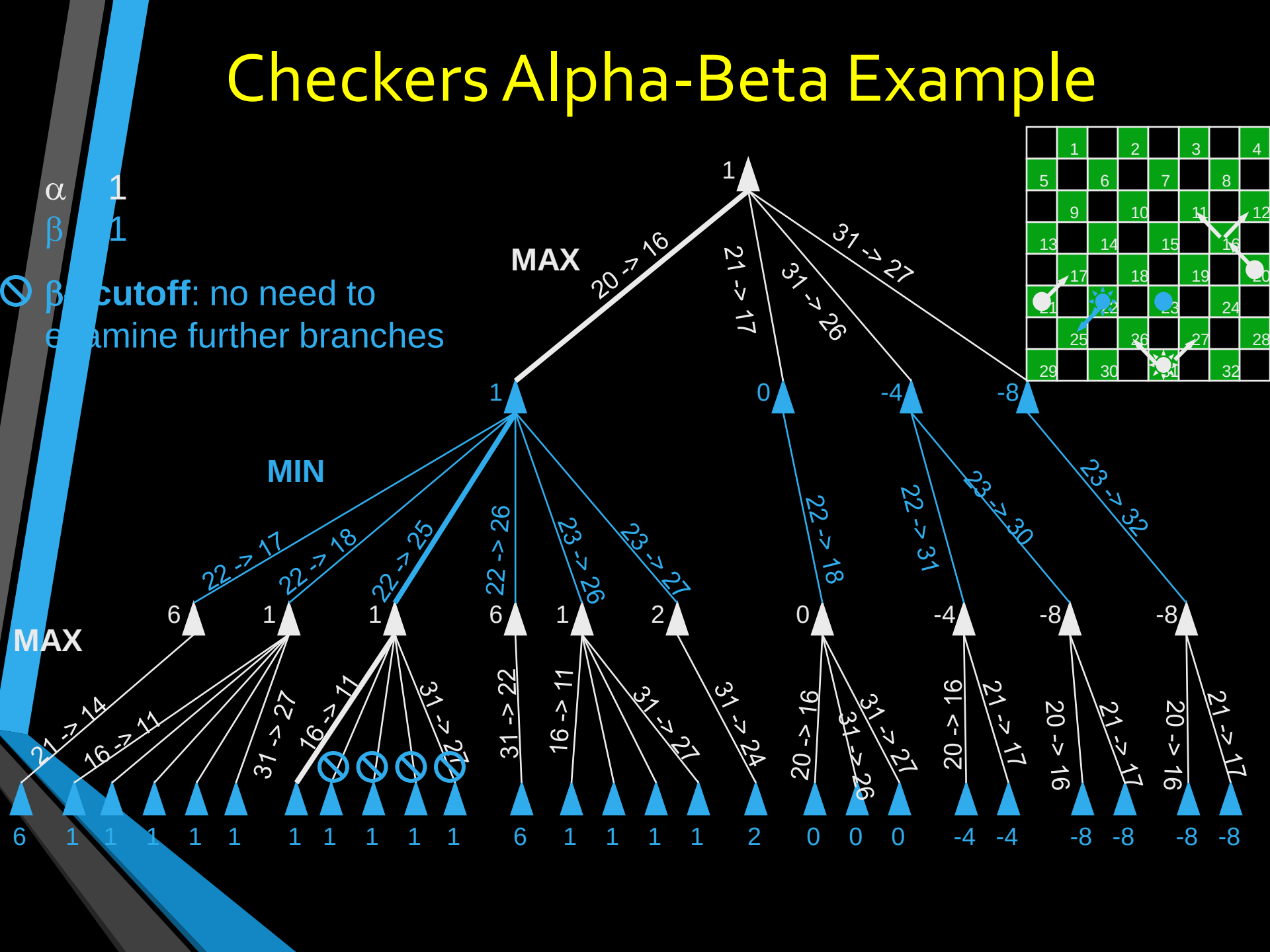
Checkers Alpha-Beta Example



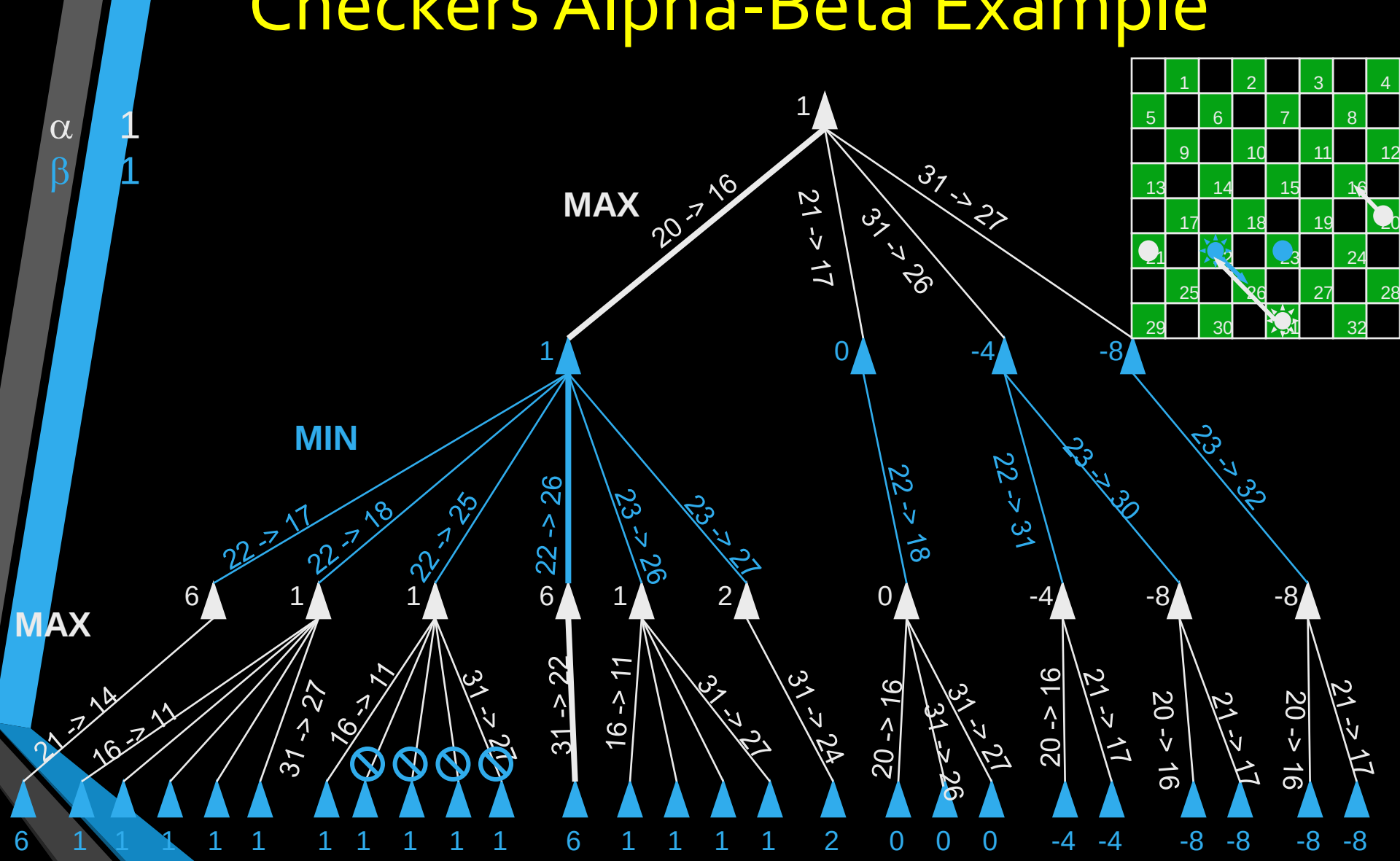
Checkers Alpha-Beta Example



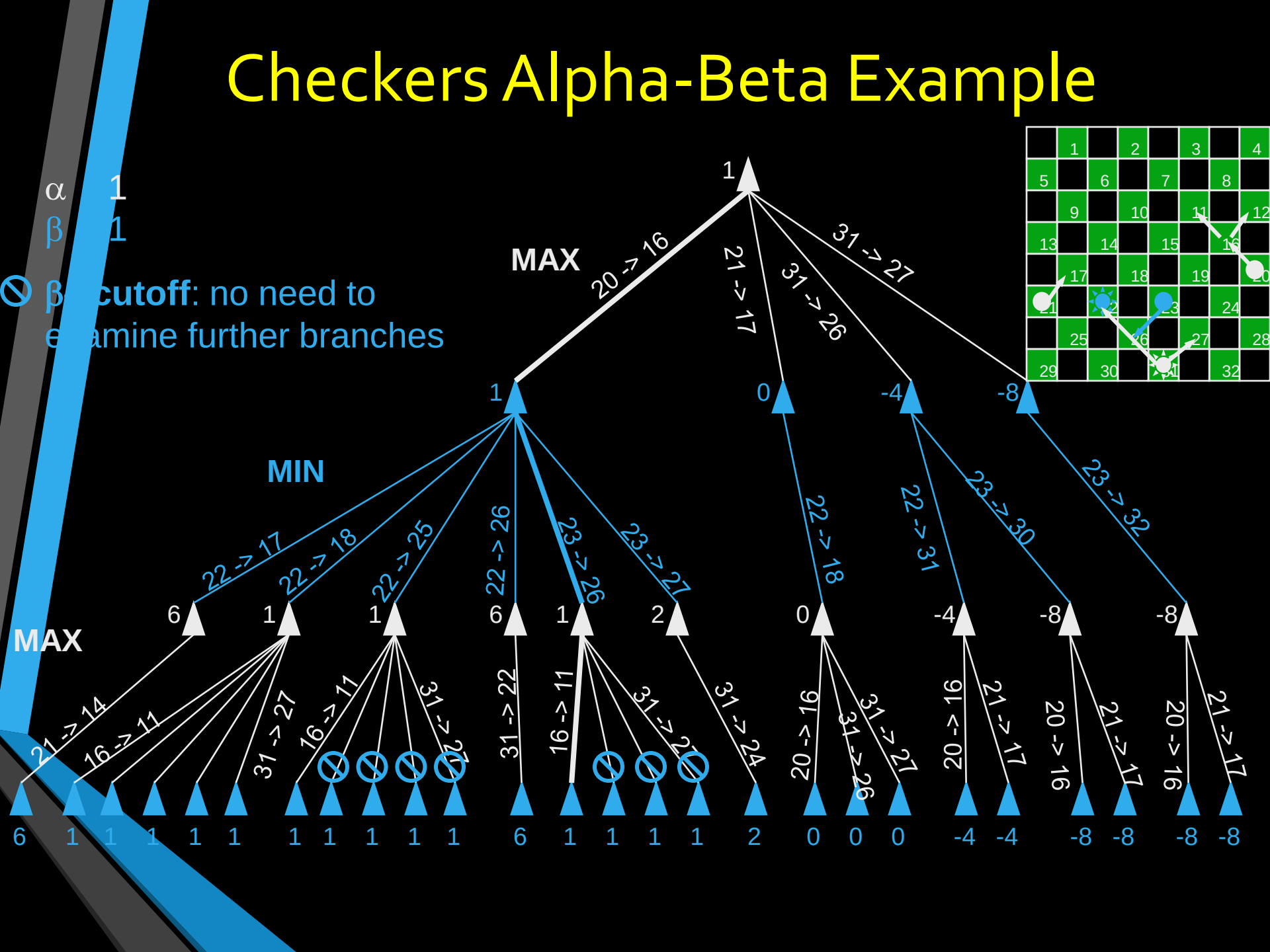
Checkers Alpha-Beta Example



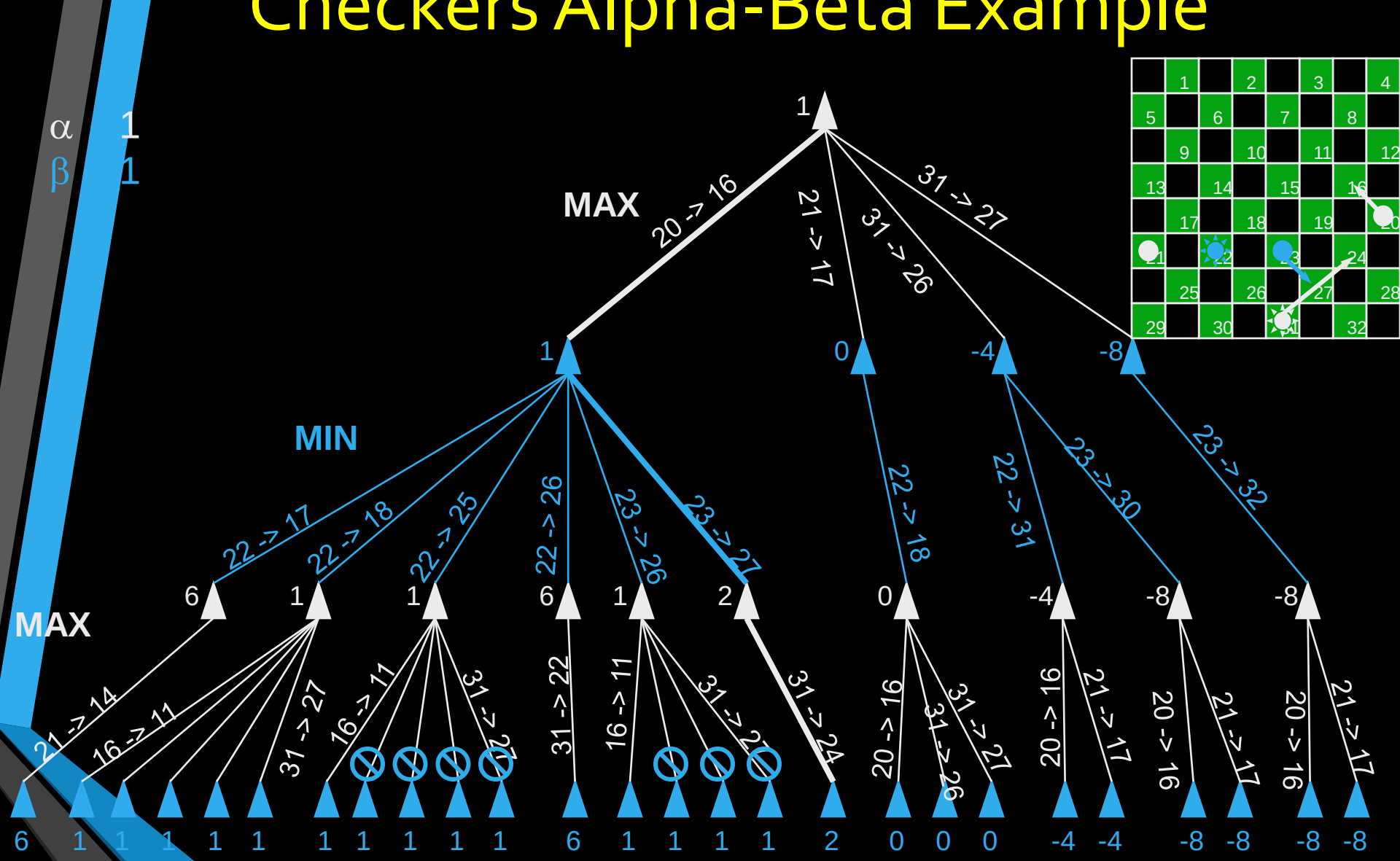
Checkers Alpha-Beta Example



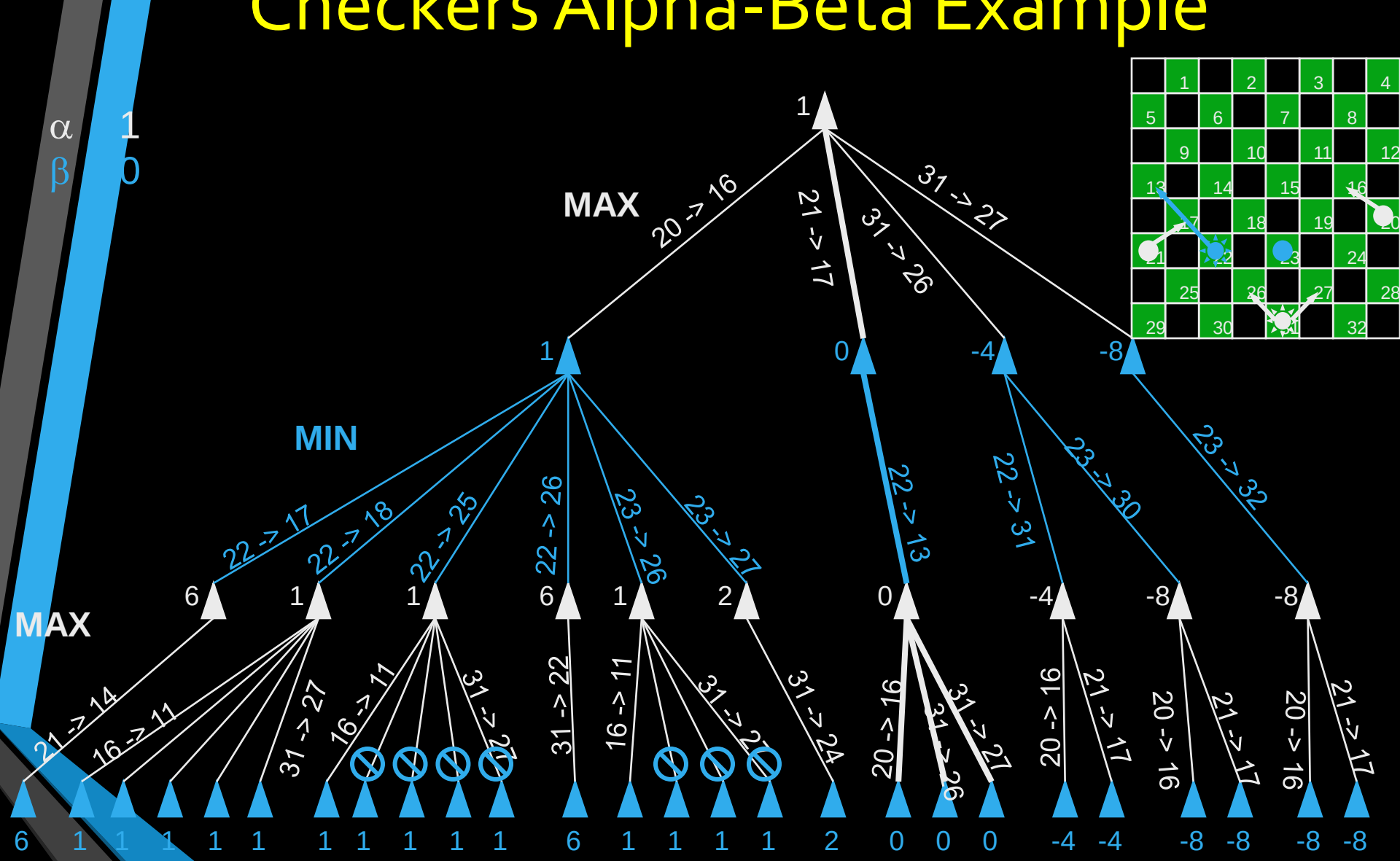
Checkers Alpha-Beta Example



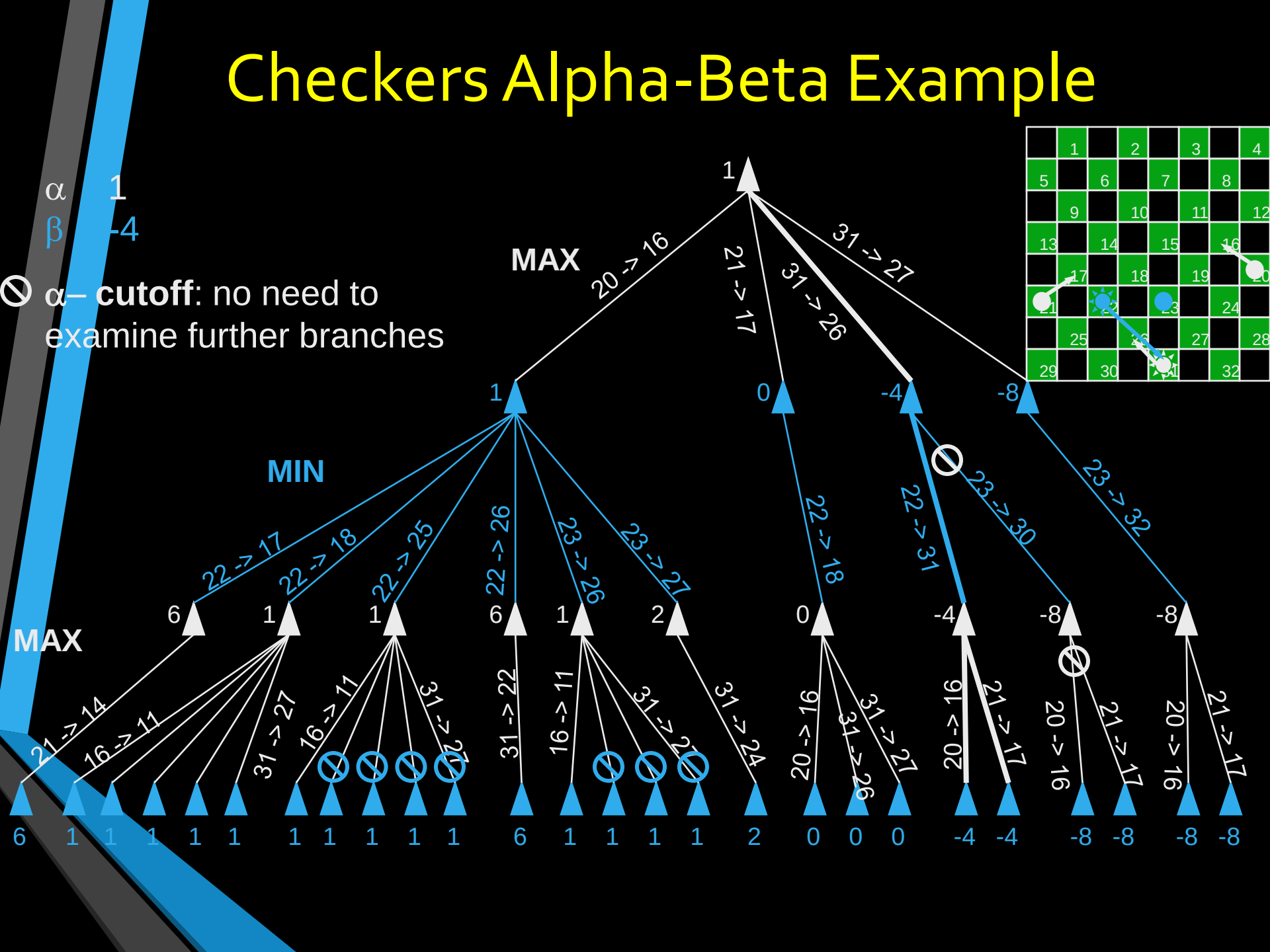
Checkers Alpha-Beta Example



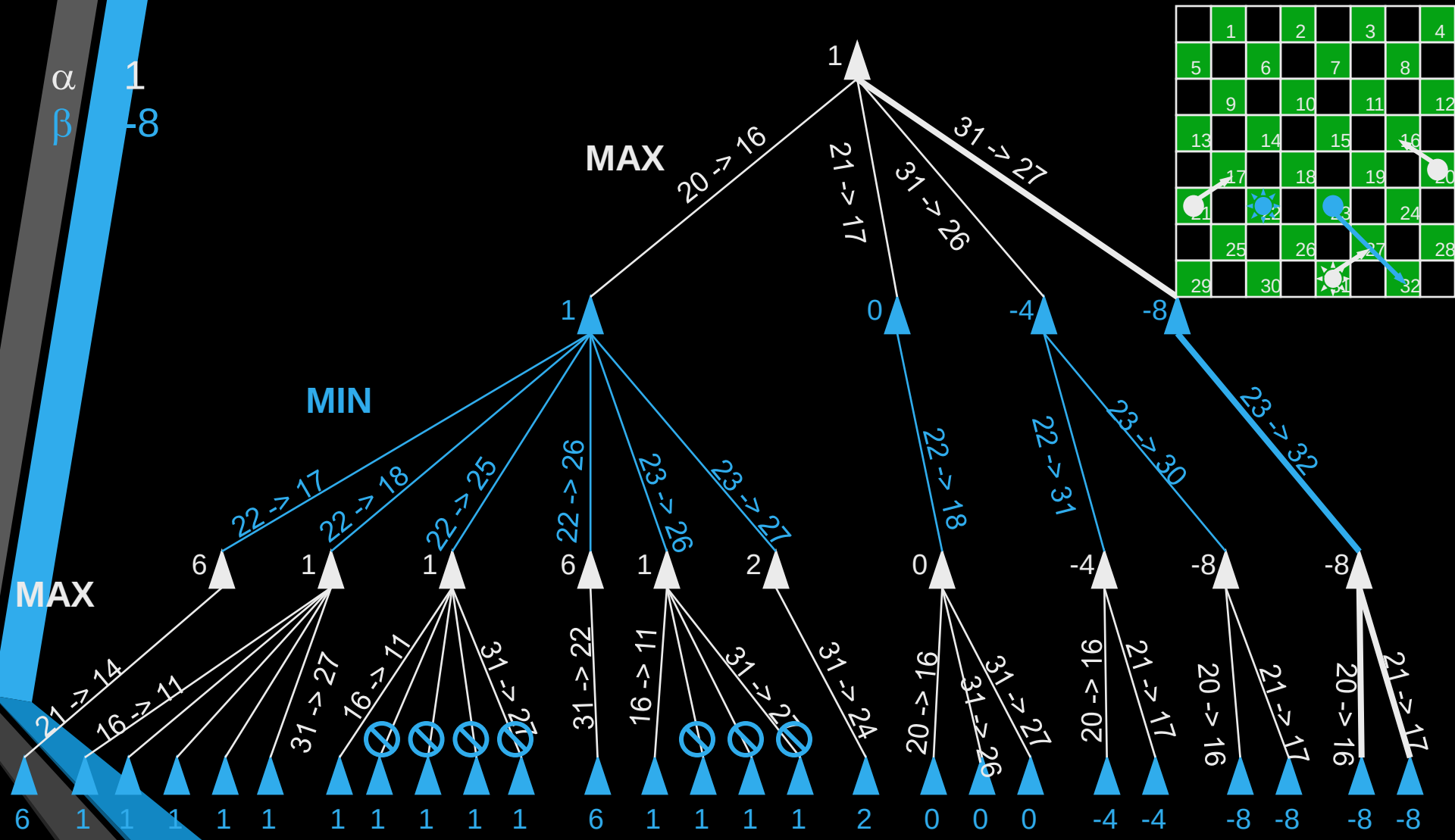
Checkers Alpha-Beta Example



Checkers Alpha-Beta Example



Checkers Alpha-Beta Example



Horizon Problem

- moves may have disastrous consequences in the future, but the consequences are not visible
  - the corresponding change in the evaluation function will only become evident at deeper levels
    - they are "beyond the horizon"
- determining the horizon is an open problem without a general solution
  - only some pragmatic approaches restricted to specific games or situation

Games and Computers

- state of the art for some game programs
  - Chess
  - Checkers
  - Othello
  - Backgammon
  - Go

Chess

- Deep Blue, a special-purpose parallel computer, defeated the world champion Gary Kasparov in 1997
  - the human player didn't show his best game
    - some claims that the circumstances were questionable
  - Deep Blue used a massive data base with games from the literature
- Fritz, a program running on an ordinary PC, challenged the world champion Vladimir Kramnik to an eight-game draw in 2002
  - top programs and top human players are roughly equal

Checkers

- Arthur Samuel develops a checkers program in the 1950s that learns its own evaluation function
  - reaches an expert level stage in the 1960s
- Chinook becomes world champion in 1994
  - human opponent, Dr. Marion Tinsley, withdraws for health reasons
    - Tinsley had been the world champion for 40 years
  - Chinook uses off-the-shelf hardware, alpha-beta search, end-games data base for six-piece positions

Othello

- Logistello defeated the human world champion in 1997
- many programs play far better than humans
  - smaller search space than chess
  - little evaluation expertise available

Backgammon

- TD-Gammon, neural-network based program, ranks among the best players in the world
  - improves its own evaluation function through learning techniques
  - search-based methods are practically hopeless
    - chance elements, branching factor

Go

- humans play far better
  - large branching factor (around 360)
    - search-based methods are hopeless
- rule-based systems play at amateur level
- the use of pattern-matching techniques can improve the capabilities of programs
  - difficult to integrate
- $2,000,000 prize for the first program to defeat a top-level player