Minimax and Alpha-Beta Reduction

Borrows from Spring 2006 CS 440 Lecture Slides

Motivation

- Want to create programs to play games
- Want to play optimally
- Want to be able to do this in a reasonable amount of time

Types of Games

	Deterministic	Nondeterministic (Chance)
Fully Observable	Chess Checkers Go Othello	Backgammon Monopoly
Partially Observable	Battleship	Card Games

Minimax is for deterministic, fully observable games

Basic Idea

- Search problem
  - Searching a tree of the possible moves in order to find the move that produces the best result
  - Depth First Search algorithm
- Assume the opponent is also playing optimally
  - Try to guarantee a win anyway!

Required Pieces for Minimax

- An initial state
  - The positions of all the pieces
  - Whose turn it is
- Operators
  - Legal moves the player can make
- Terminal Test
  - Determines if a state is a final state
- Utility Function

Utility Function

- Gives the utility of a game state
  - utility(State)
- Examples
  - -1, 0, and +1, for Player 1 loses, draw, Player 1 wins, respectively
  - Difference between the point totals for the two players
  - Weighted sum of factors (e.g. Chess)
    - $utility(S) = w_1 f_1(S) + w_2 f_2(S) + \ldots + w_n f_n(S)$
      - $f_1(S)$ = (Number of white queens) – (Number of black queens), $w_1 = 9$
      - $f_2(S)$ = (Number of white rooks) – (Number of black rooks), $w_2 = 5$
      - ...

Two Agents

- MAX
  - Wants to maximize the result of the utility function
  - Winning strategy if, on MIN's turn, a win is obtainable for MAX for all moves that MIN can make
- MIN
  - Wants to minimize the result of the evaluation function
  - Winning strategy if, on MAX's turn, a win is obtainable for MIN for all moves that MAX can make

Basic Algorithm

```
function MINIMAX-DECISION(state) returns an action
   inputs: state, current state in game
   return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

```
function MAX-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← −∞
   for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
   return v
```

```
function MIN-VALUE(state) returns a utility value
   if TERMINAL-TEST(state) then return UTILITY(state)
   v ← ∞
   for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
   return v
```

Example

- Coins game
  - There is a stack of N coins
  - In turn, players take 1, 2, or 3 coins from the stack
  - The player who takes the last coin loses

Coins Game: Formal Definition

- Initial State: The number of coins in the stack
- Operators:
  1. Remove one coin
  2. Remove two coins
  3. Remove three coins
- Terminal Test: There are no coins left on the stack
- Utility Function: F(S)
  - F(S) = 1 if MAX wins, 0 if MIN wins

MAX

MIN

N = 4
K =

1 2 3

N = 3
K =

N = 2
K =

N = 1
K =

3 2 1

2 1

1

N = 0
K =

F(S)=1

N = 1
K =

N = 2
K =

N = 0
K =

F(S)=1

N = 1
K =

N = 0
K =

F(S)=1

1

1 2

1

N = 3
K =

F(S)=0

N = 1
K =

N = 0
K =

F(S)=0

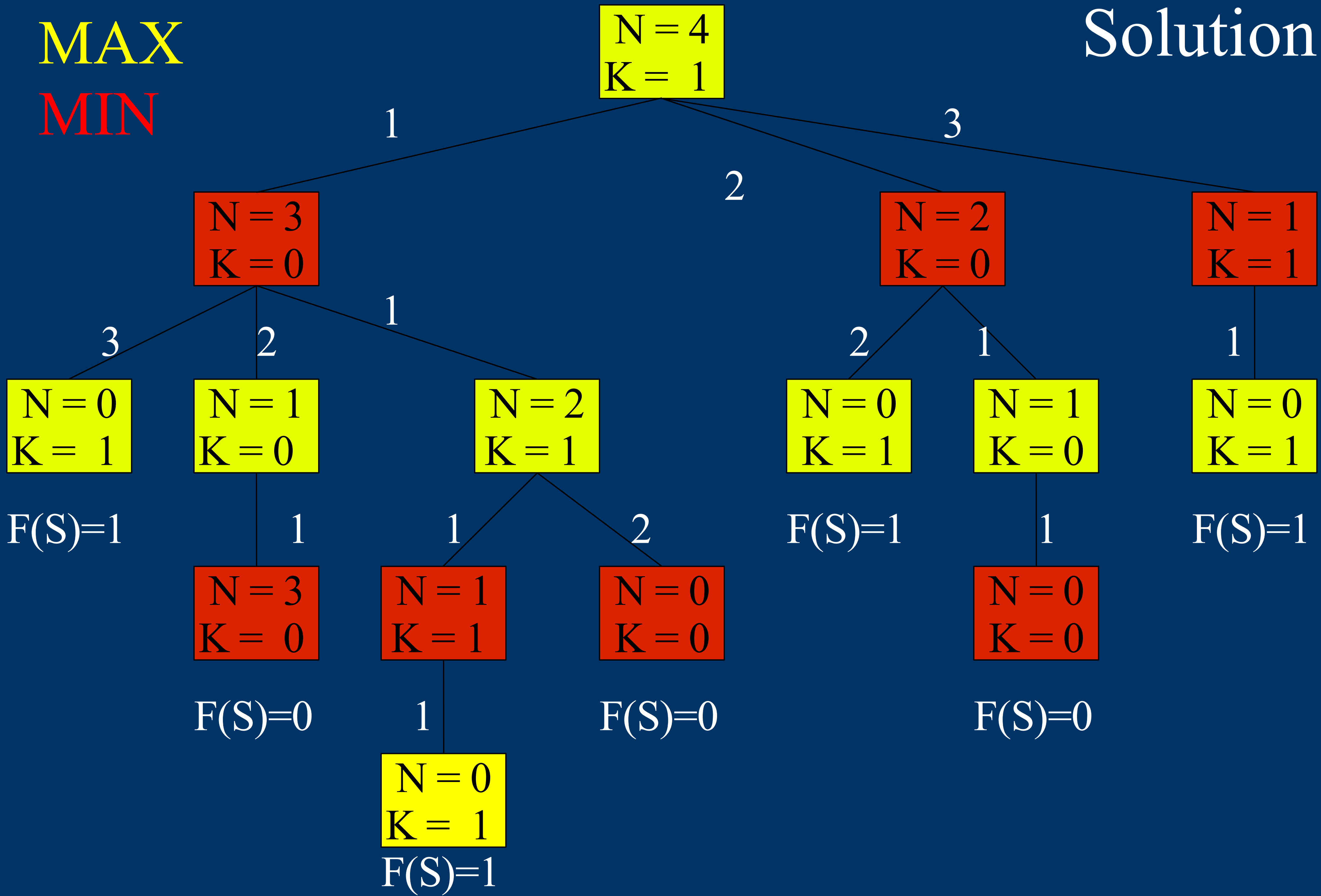
N = 0
K =

F(S)=0

1

N = 0
K =

F(S)=1


MAX
MIN
Solution
N = 4
K = 1
1
2
3
N = 3
K = 0
N = 2
K = 0
N = 1
K = 1
3
2
1
2
1
1
N = 0
K = 1
N = 1
K = 0
N = 2
K = 1
N = 0
K = 1
N = 1
K = 0
N = 0
K = 1
F(S)=1
1
1
2
F(S)=1
1
F(S)=1
N = 3
K = 0
N = 1
K = 1
N = 0
K = 0
N = 0
K = 0
F(S)=0
1
F(S)=0
F(S)=0
N = 0
K = 1
F(S)=1

Analysis

- Max Depth: 5
- Branch factor: 3
- Number of nodes: 15
- Even with this trivial example, you can see that these trees can get very big
  - Generally, there are $O(b^d)$ nodes to search for
    - Branch factor b: maximum number of moves from each node
    - Depth d: maximum depth of the tree
  - Exponential time to run the algorithm!
  - How can we make it faster?

Alpha-Beta Pruning

- Main idea: Avoid processing subtrees that have no effect on the result
- Two new parameters
  - α: The best value for MAX seen so far
  - β: The best value for MIN seen so far
- α is used in MIN nodes, and is assigned in MAX nodes
- β is used in MAX nodes, and is assigned in MIN nodes

Alpha-Beta Pruning

- MAX (Not at level 0)
  - If a subtree is found with a value k greater than the value of β, then we do not need to continue searching subtrees
    - MAX can do at least as good as k in this node, so MIN would never choose to go here!
- MIN
  - If a subtree is found with a value k less than the value of α, then we do not need to continue searching subtrees
    - MIN can do at least as good as k in this node, so MAX would never choose to go here!

Algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(a, *state*))

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

MAX

MIN

N = 4
K =
α =
β =

1

2

3

α =
β =

N = 3
K =
α =
β =

N = 2
K =
α =
β =

N = 1
K =

α =
β =

3

2

1

α =
β =

2

1

1

N = 0
K =

N = 1
K =
α =
β =

N = 2
K =
α =
β =

N = 0
K =

N = 1
K =
α =
β =

N = 0
K =

F(S)=1

1

1

2

F(S)=1

1

F(S)=1

α =
β =

N = 3
K =

N = 1
K =
α =
β =

N = 0
K =
α =
β =

N = 0
K =
α =
β =

α
=
β
=

F(S)=0

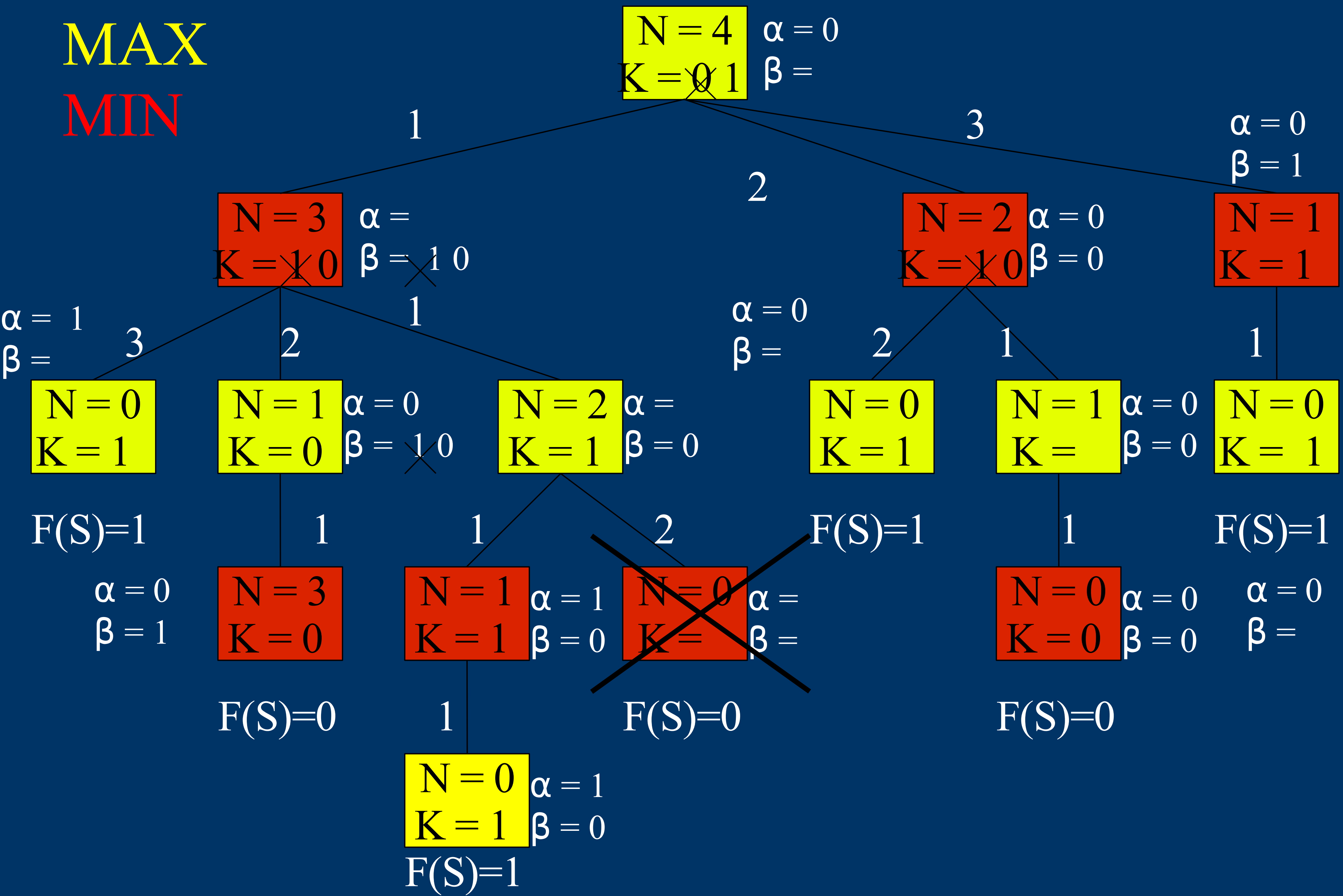
1

F(S)=0

F(S)=0

N = 0
K =
α =
β =

F(S)=1

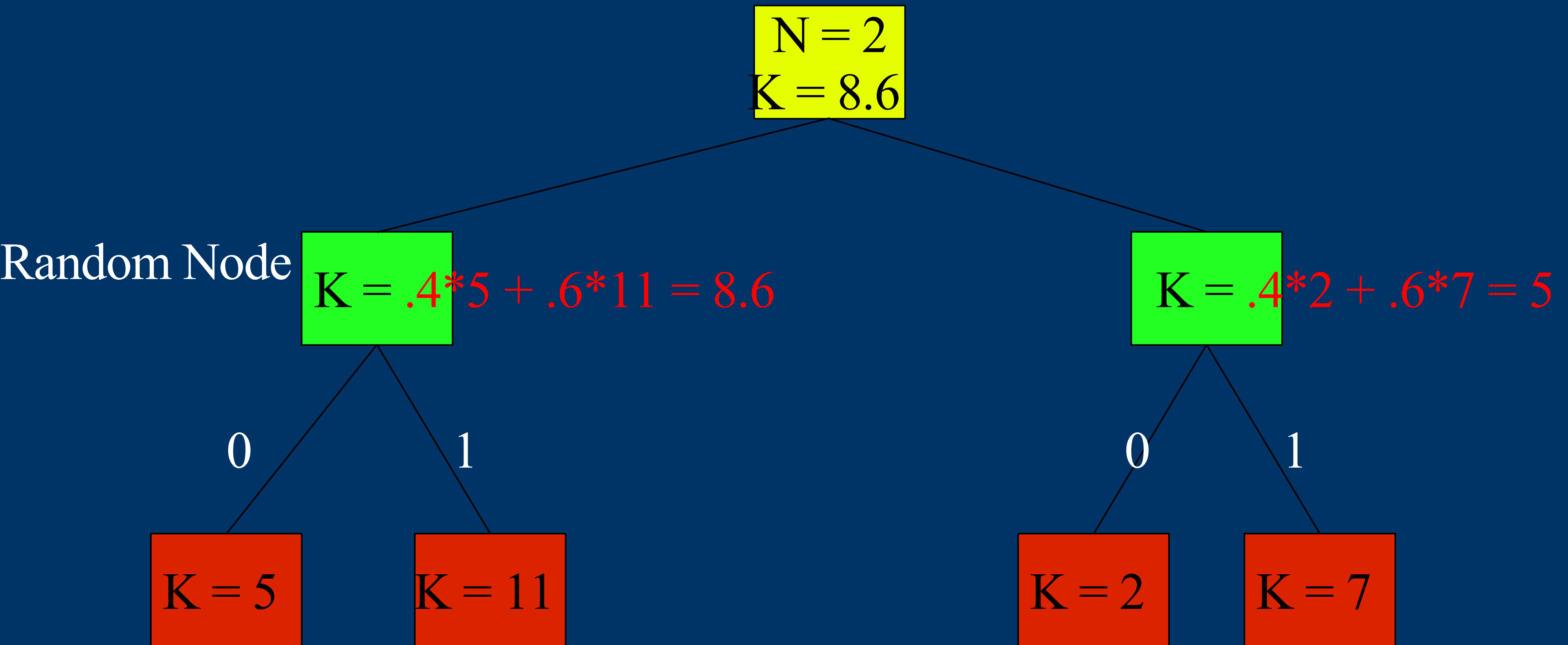


MAX
MIN
N = 4
K = 0 1
α = 0
β =
1
2
3
N = 3
K = 1 0
α =
β = 1 0
N = 2
K = 1 0
α = 0
β = 0
α = 0
β = 1
N = 1
K = 1
α = 1
β =
3
2
1
α = 0
β =
2
1
1
N = 0
K = 1
N = 1
K = 0
α = 0
β = 1 0
N = 2
K = 1
α =
β = 0
N = 0
K = 1
N = 1
K =
α = 0
β = 0
N = 0
K = 1
F(S)=1
1
1
2
F(S)=1
1
F(S)=1
α = 0
β = 1
N = 3
K = 0
N = 1
K = 1
α = 1
β = 0
N = 0
K =
α =
β =
N = 0
K = 0
α = 0
β = 0
α = 0
β =
F(S)=0
1
F(S)=0
F(S)=0
N = 0
K = 1
α = 1
β = 0
F(S)=1

Nondeterministic Games

- Minimax can also be used for nondeterministic games (those that have an element of chance)
- There is an additional node added (Random node)
- Random node is between MIN and MAX (and vice versa)
- Make subtrees over all of the possibilities,and average the results

Example



Our Project

- We will focus on deterministic, two-player, fully observable games
- We will be trying to learn the evaluator function, in order to save time when playing the game
  - Training on data from Minimax runs (Neural Network)
  - Having the program play against itself (Genetic Algorithms)

Conclusion

- Minimax finds optimal play for deterministic, fully observable, two-player games
- Alpha-Beta reduction makes it faster