Mini Project 3
Gomoku AI

Due: 6/22 (Wed.)

Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

Outline

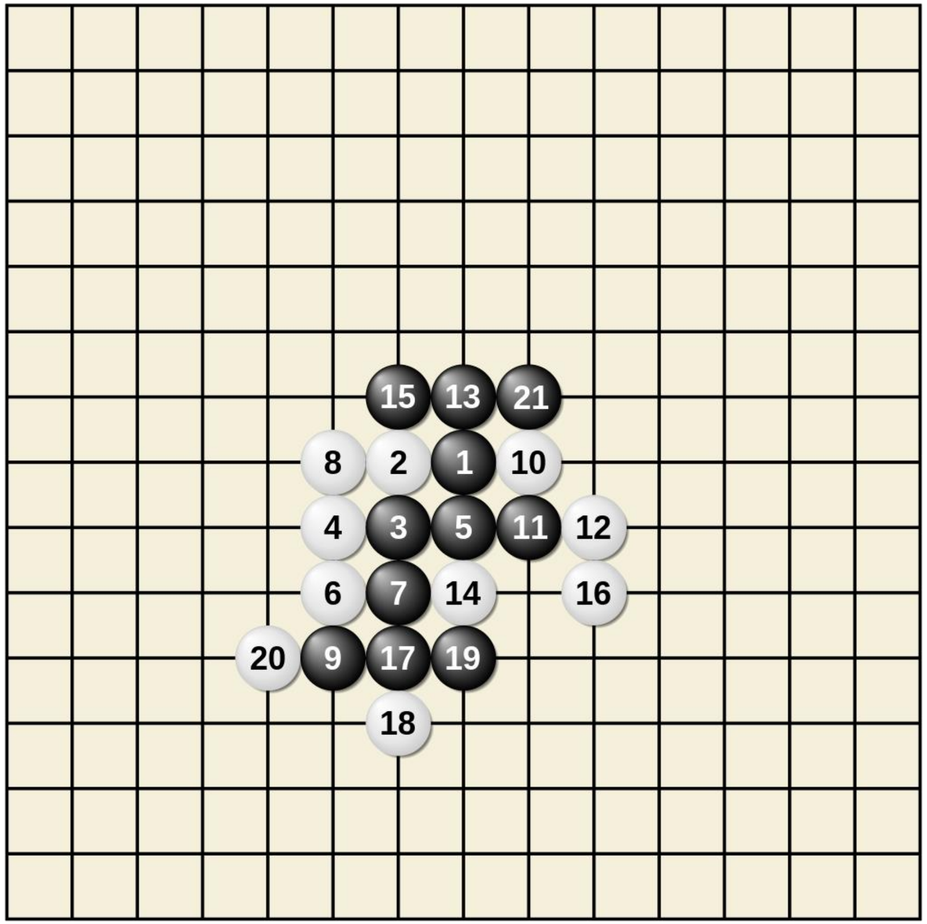
Introduction

- Design and implement an AI that can play the boardgame Gomoku
- Read the current board and output the next move
- Design a state value function to evaluate the score of the board
- Determine the next move with tree search algorithm

Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

Gomoku (Five in a Row)



Source: https://en.wikipedia.org/wiki/Gomoku

Gomoku Rules

- Use black and white discs and an 15 x 15 board, black plays first
- Starts with initial empty board
- Player can place disc at any empty intersection
- The winner is the first player to form an unbroken chain of five stones (horizontally, vertically, or diagonally).

Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

State Value Function

- The program should decide which move is better
- We can pick the move which leads to the board with highest score
- Thus, we need a function to evaluate the score of the board
- It is the “state value function”

State Value Function

- State => the board

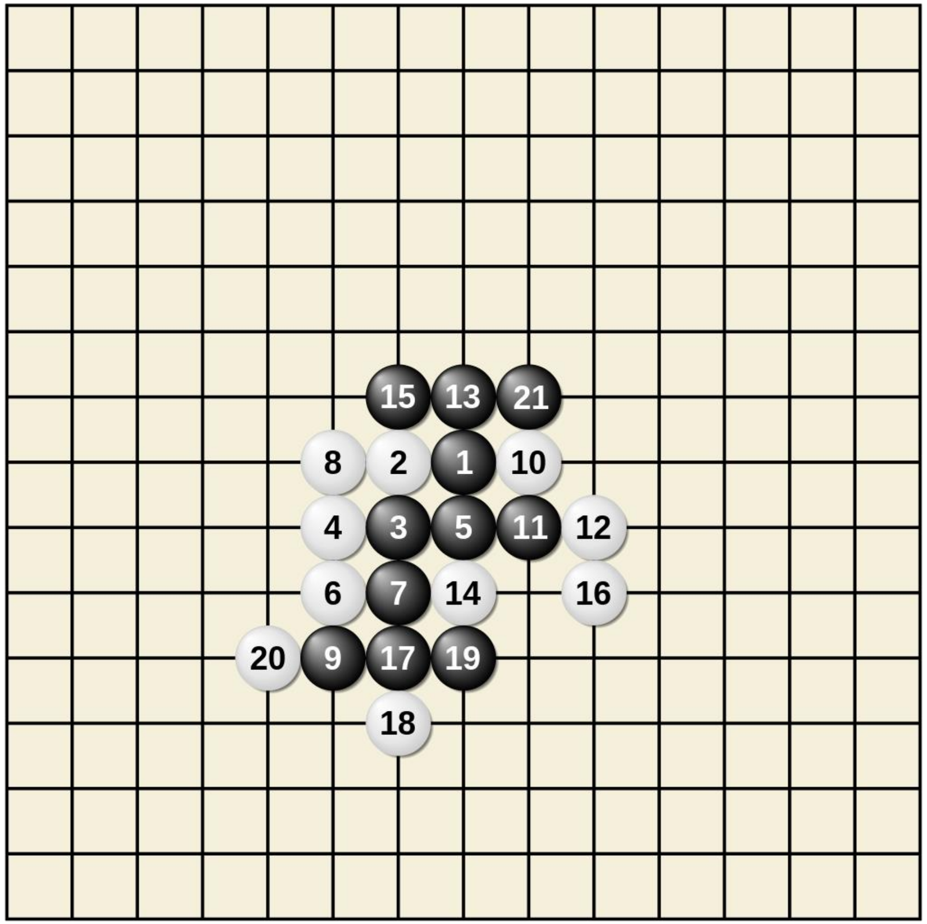
- Value => how “good” the board is

- Function => given a board, output the value

Simple Example

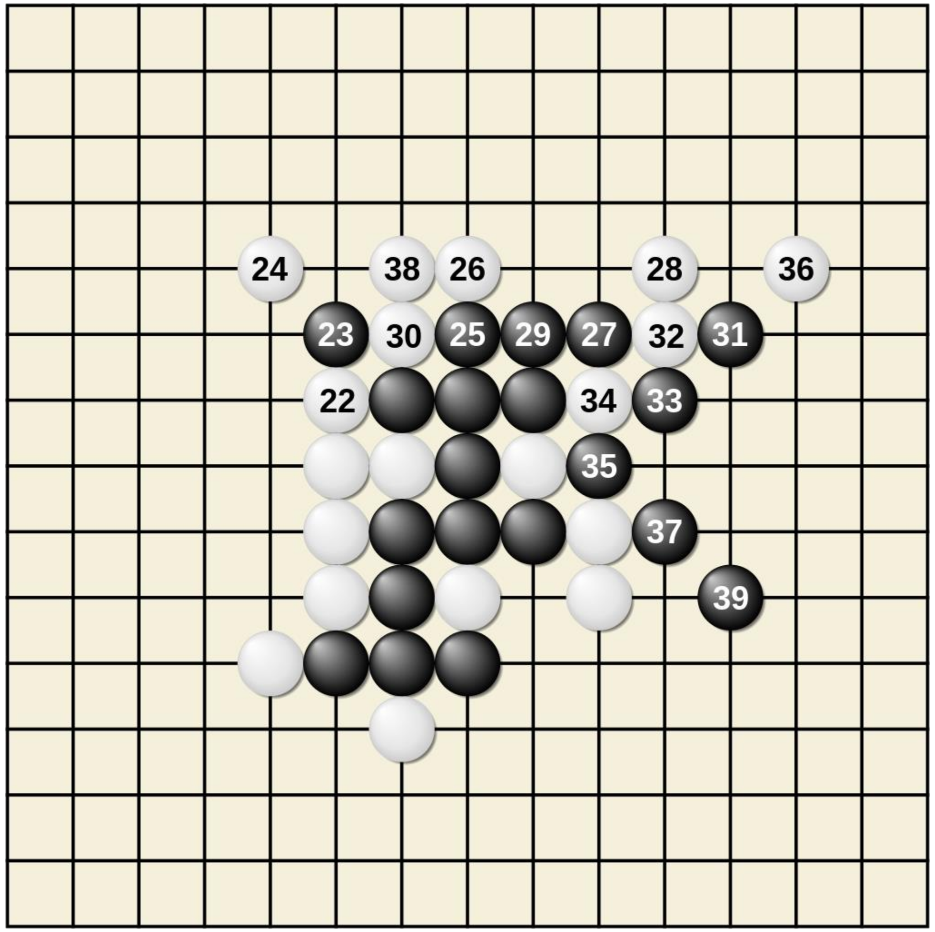
- According to the Gomoku rule, we win if we have five in a row
- Thus, we can define a simple value function:
- Value = # of 3, 4 and 5 continuous player's discs
  -# of 4 and 5 continuous opponent's discs

Suppose we play as black, value = 8 – 0 = 8



Source: https://en.wikipedia.org/wiki/Gomoku

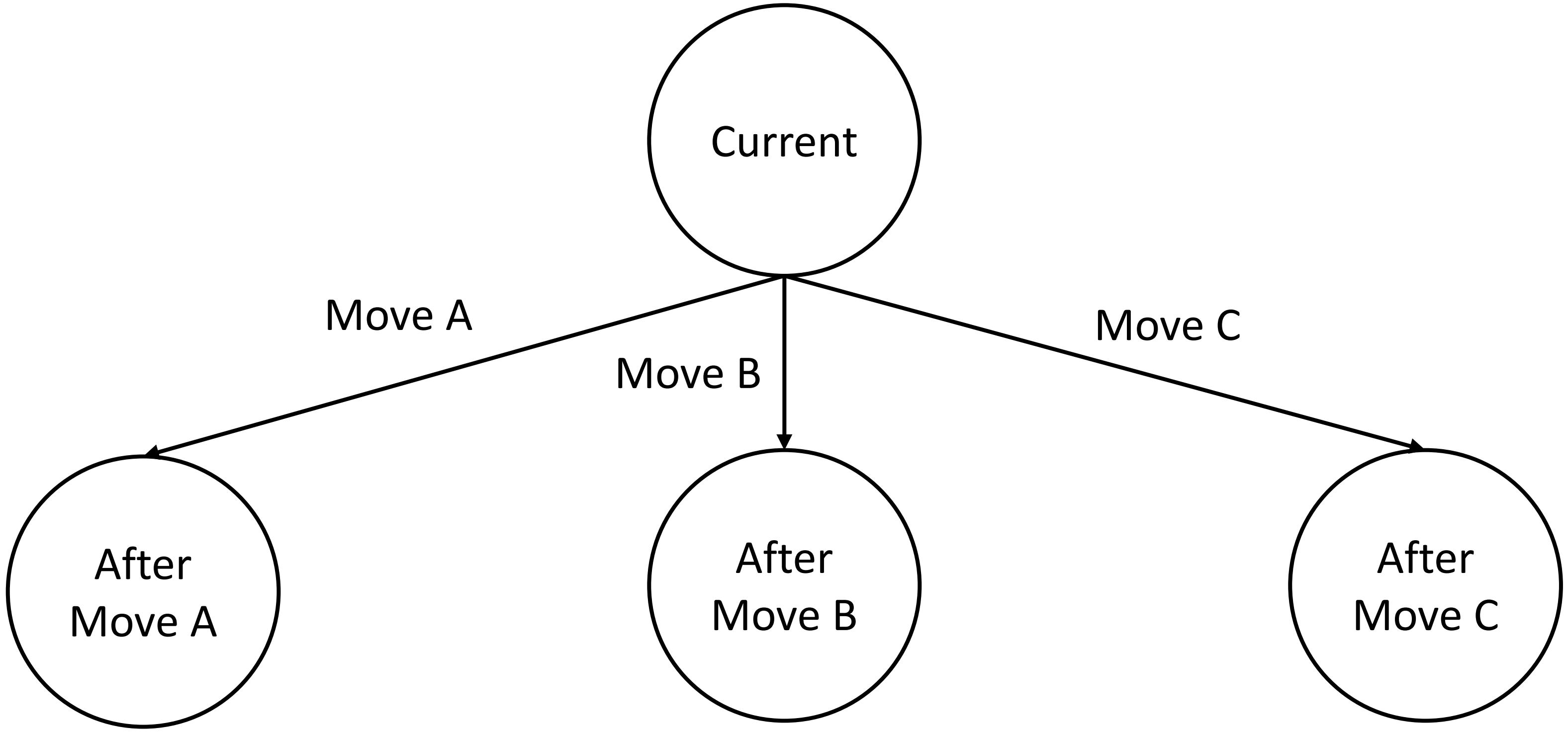
Another example, value = 11 – 1 = 10



Source: https://en.wikipedia.org/wiki/Gomoku

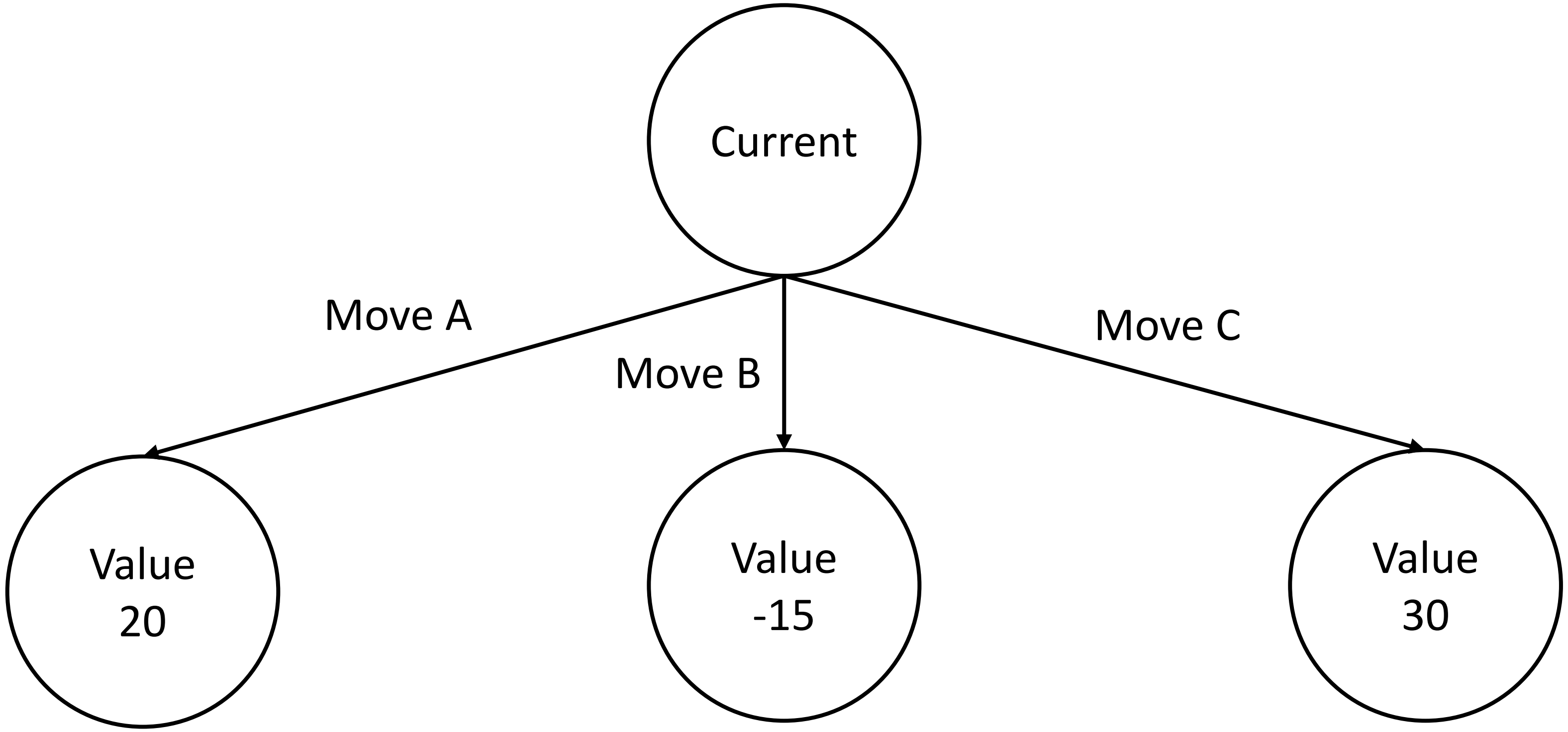
Use value function to pick the next move

- Suppose we have three valid moves, A, B, and C:



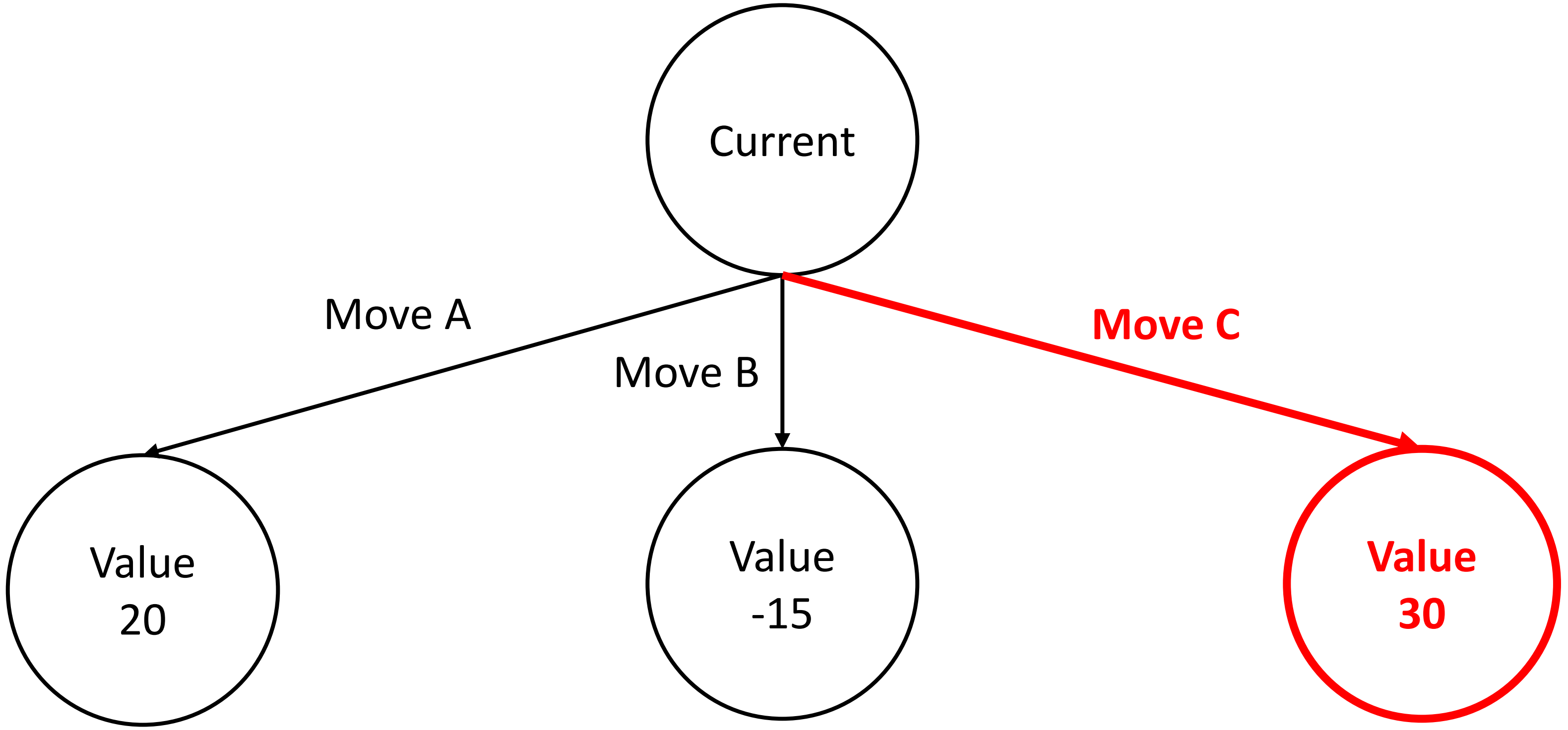
Use value function to pick the next move

- After evaluating the state values, we have 20, -15, and 30



Use value function to pick the next move

- We pick move C to be our next step since it leads to the highest value



Stronger Value Functions

- The value function in the previous example is too simple
- It is nearly impossible to beat the baselines by using it
- In this mini project, you must design a stronger value function to evaluate how “good” the board is

Features You Can Utilize

- Disc count
- Valid move count
- Disc position
- Game status (win, lose)
- Try to figure out more features by yourself

Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

Minimax

- In the previous example, we only look forward for one step
- However, the opponent will try its best to defeat you
- Greedy choice is not always the best
- We should look forward for more steps and simulate how the opponent thinks to make the best choice with least risk

Minimax

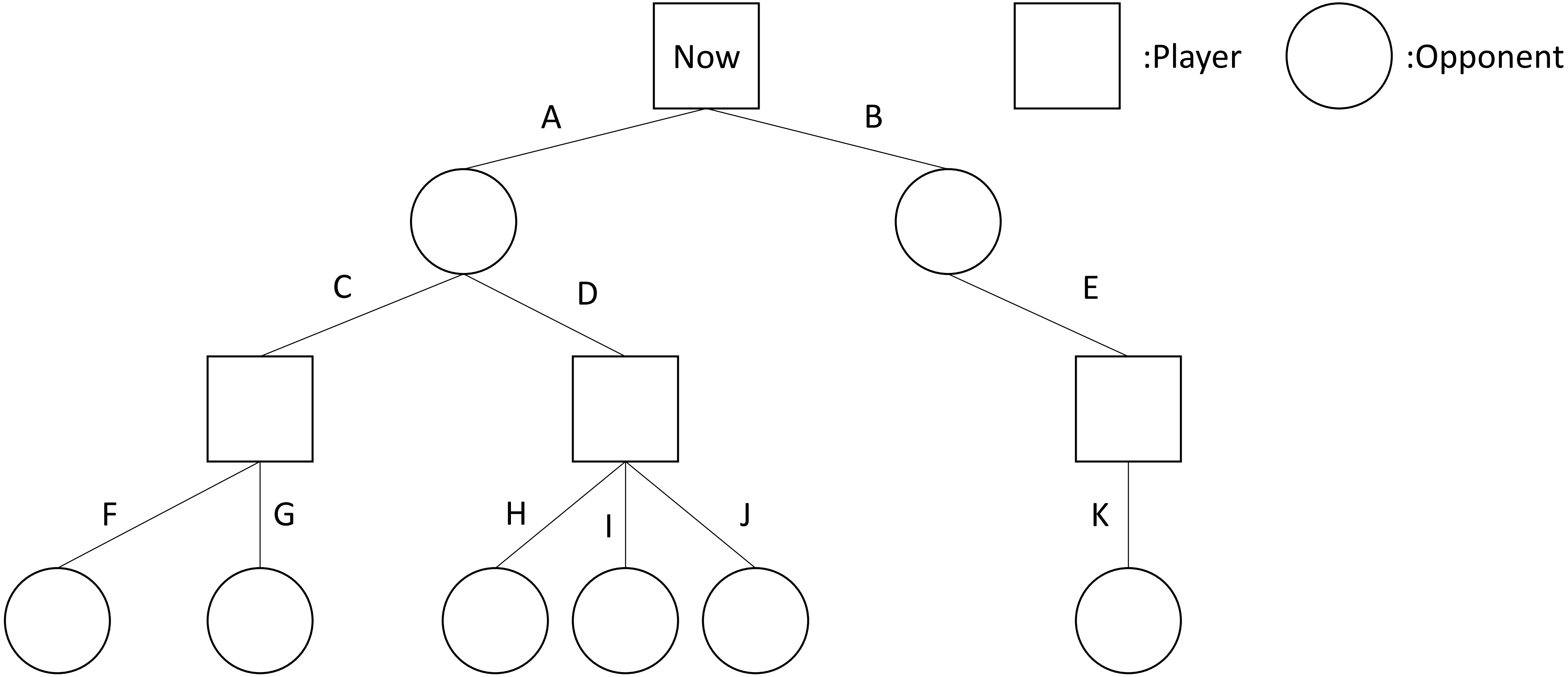
- Player tries its best to win
  - Player picks the move with the highest score
- Opponent tries its best to defeat the player
  - Opponent picks the move with the lowest "player's value function" score
  - That is, opponent tends to give the player the worst board
- The Minimax algorithm is based on this player-opponent interaction

Minimax Pseudocode

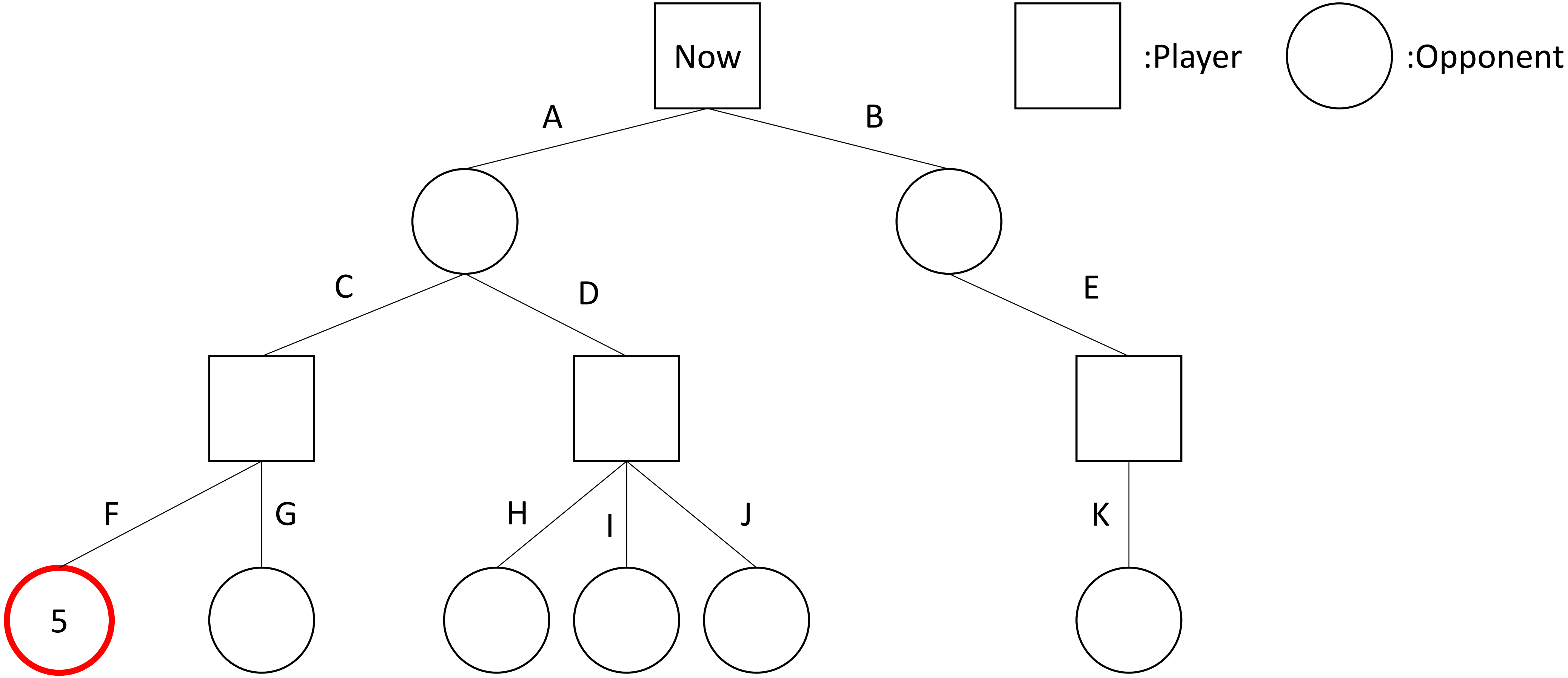
```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

Source: https://en.wikipedia.org/wiki/Minimax

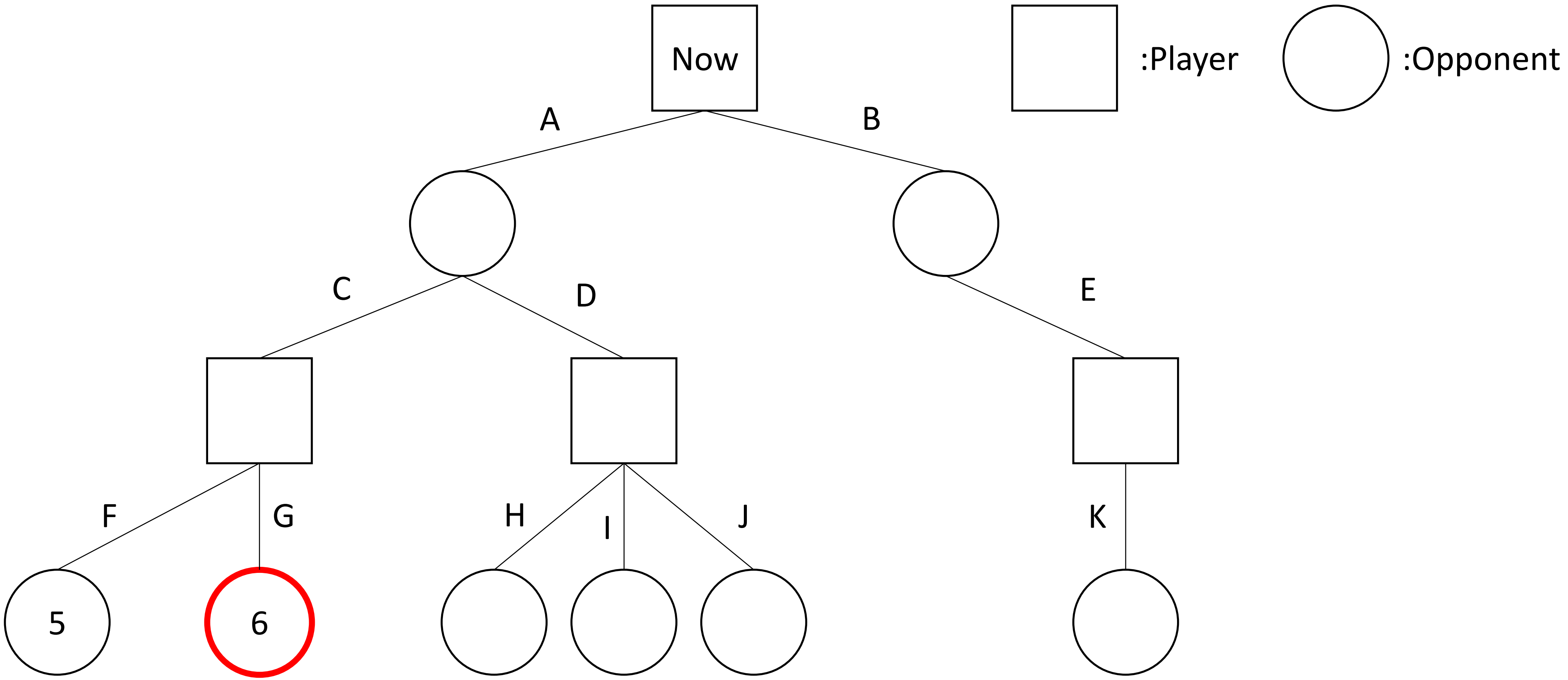
Example



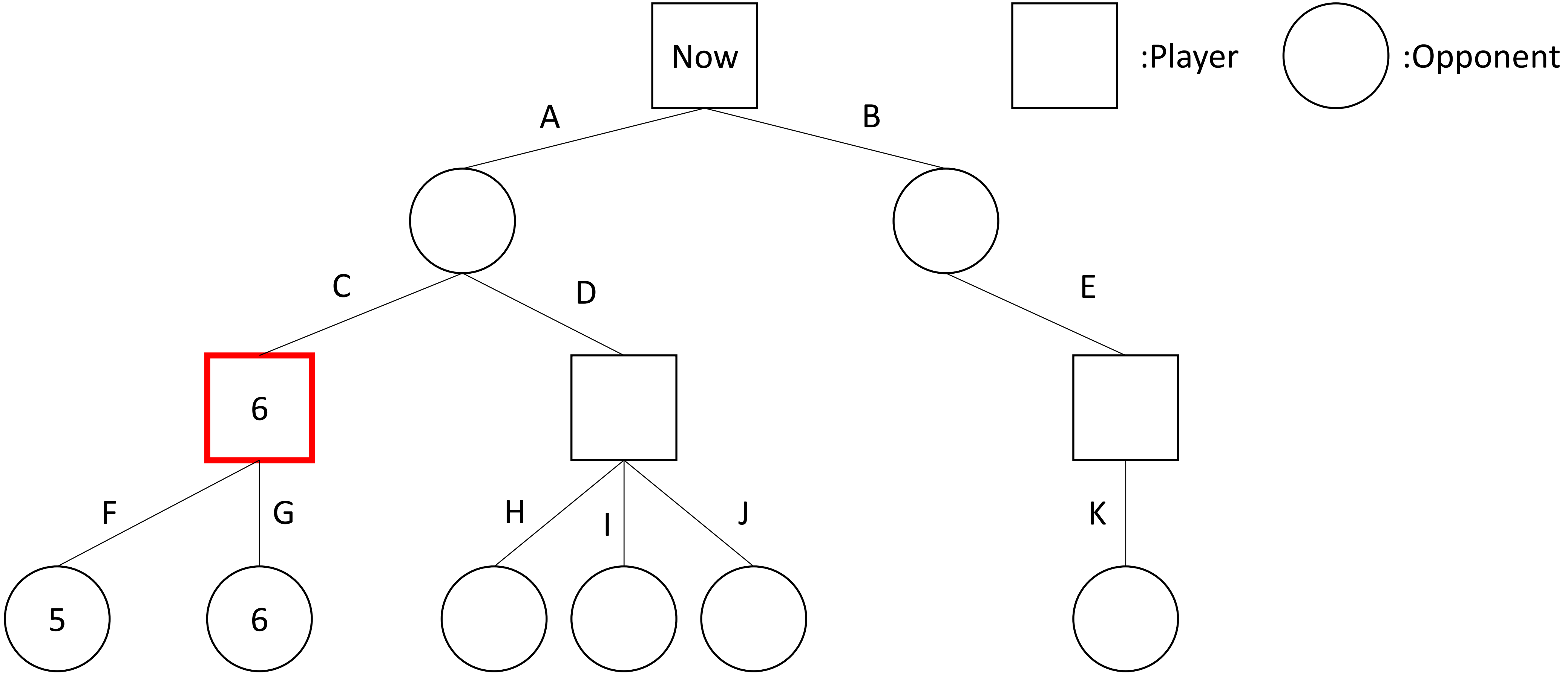
Evaluate score at leaves



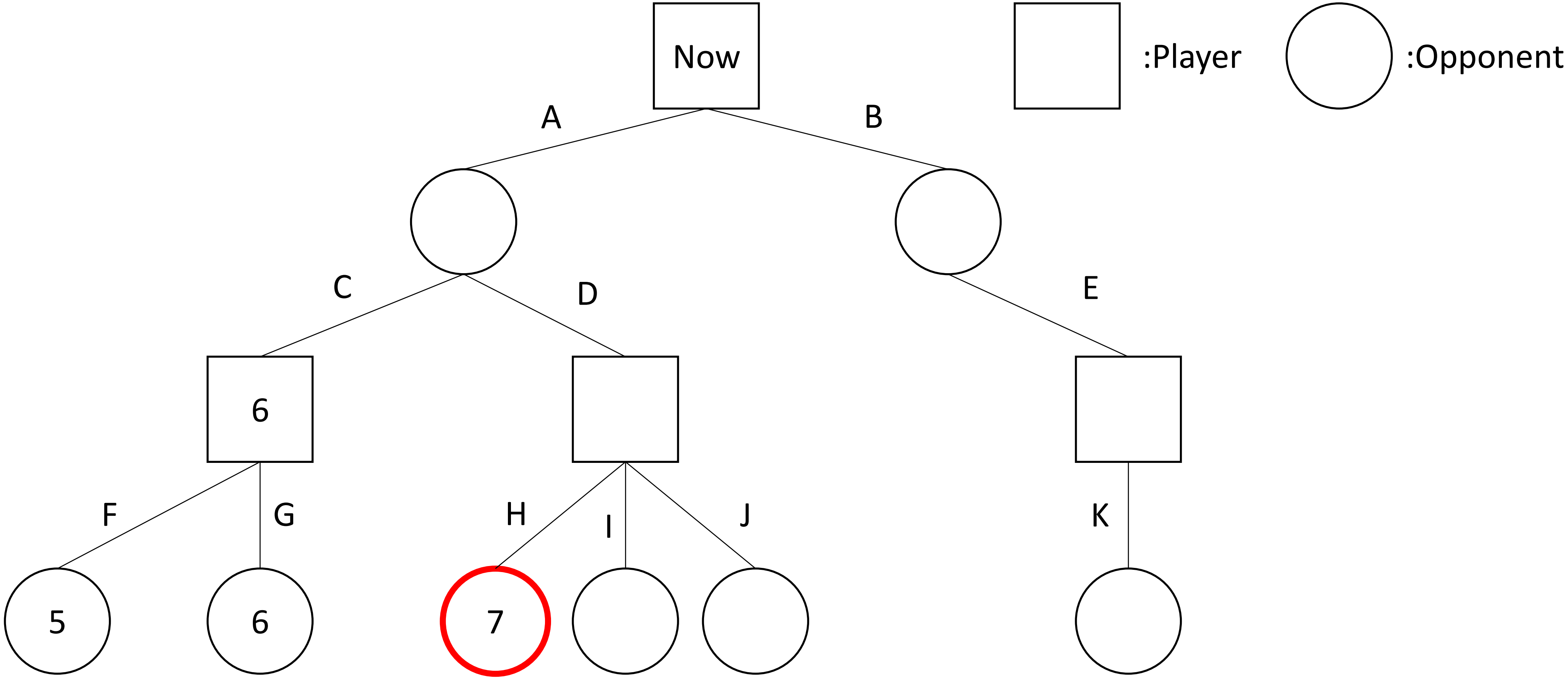
Evaluate score at leaves



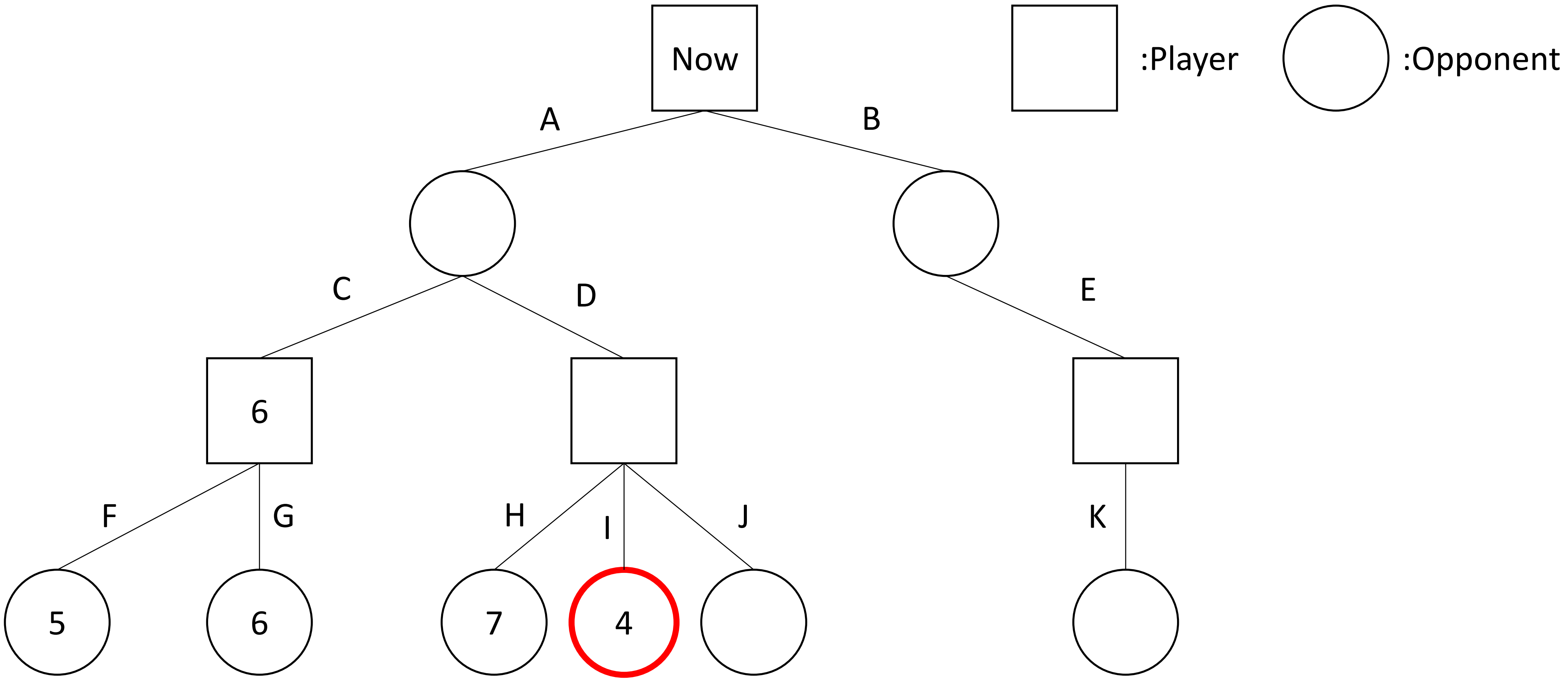
Player picks the largest score



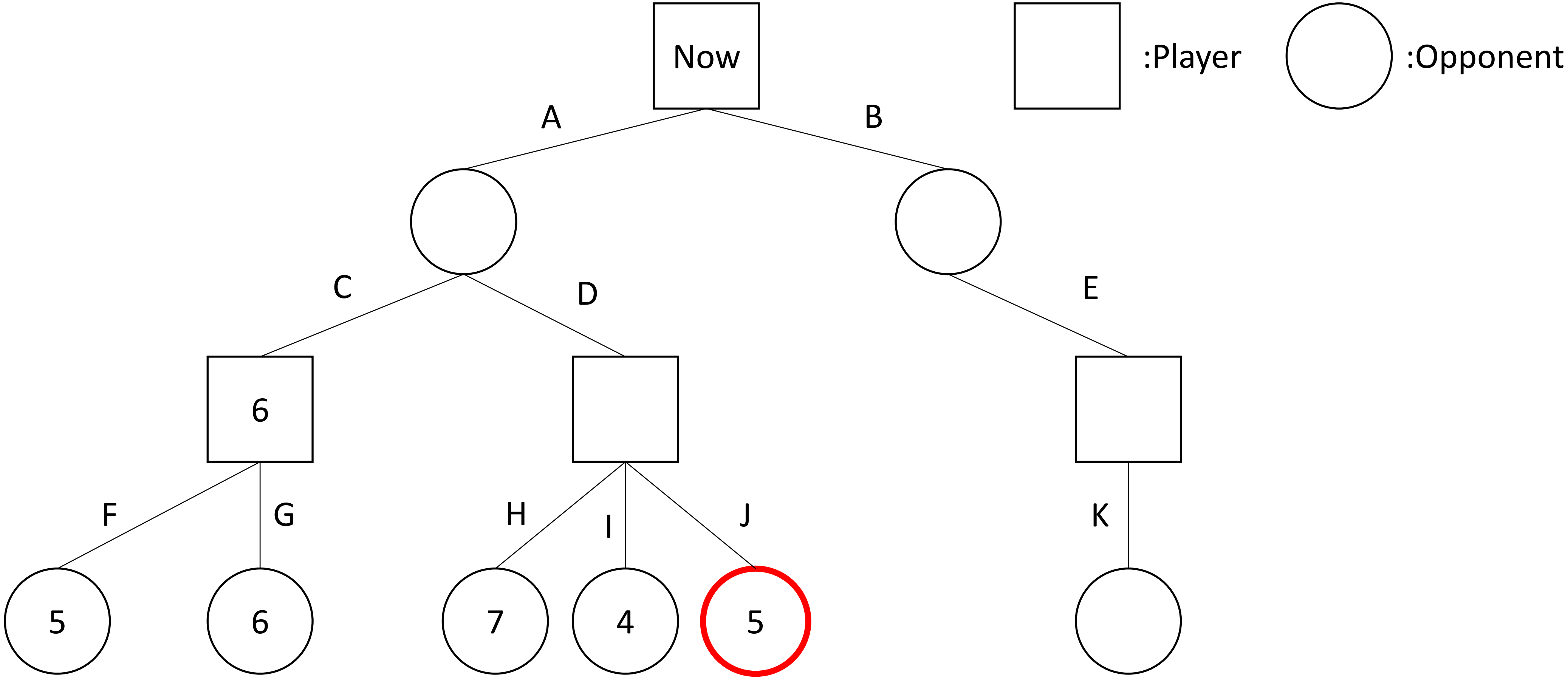
Evaluate score at leaves



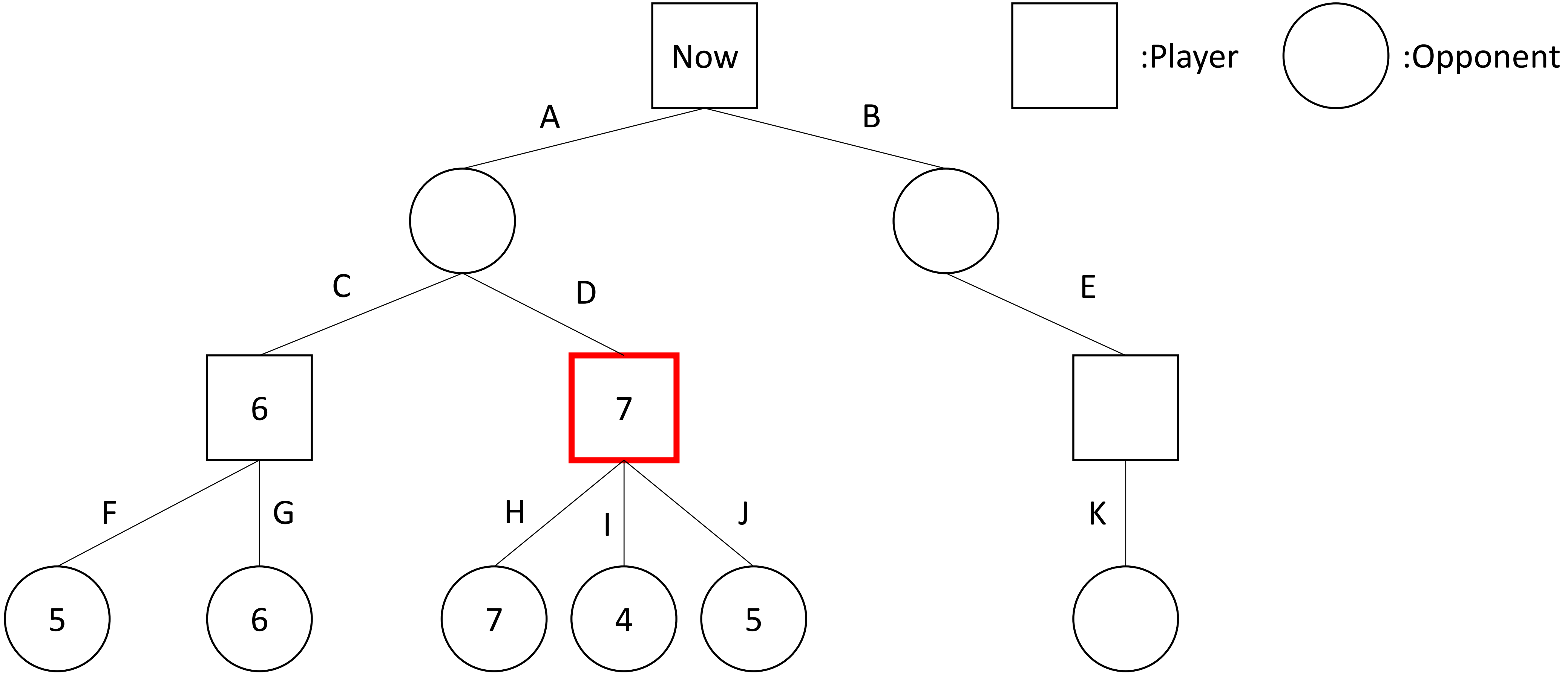
Evaluate score at leaves



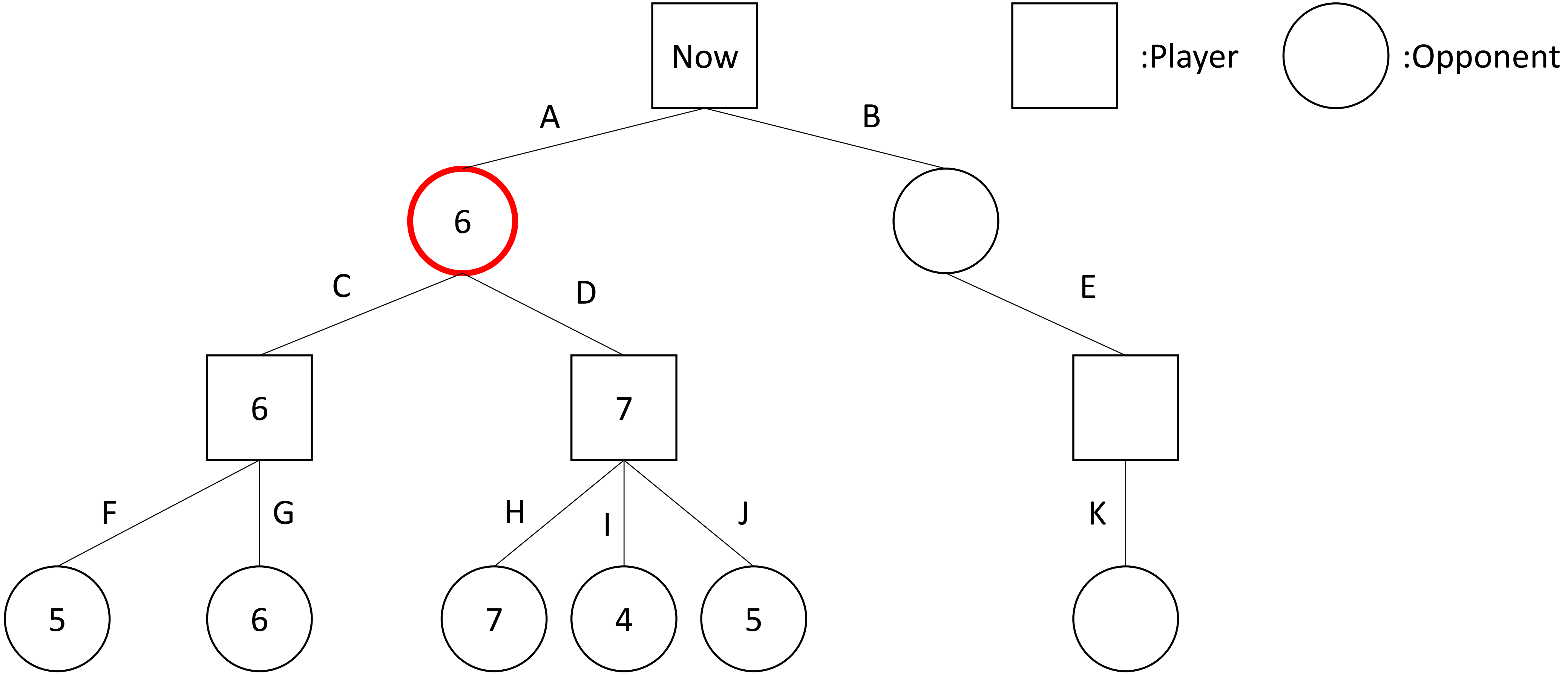
Evaluate score at leaves



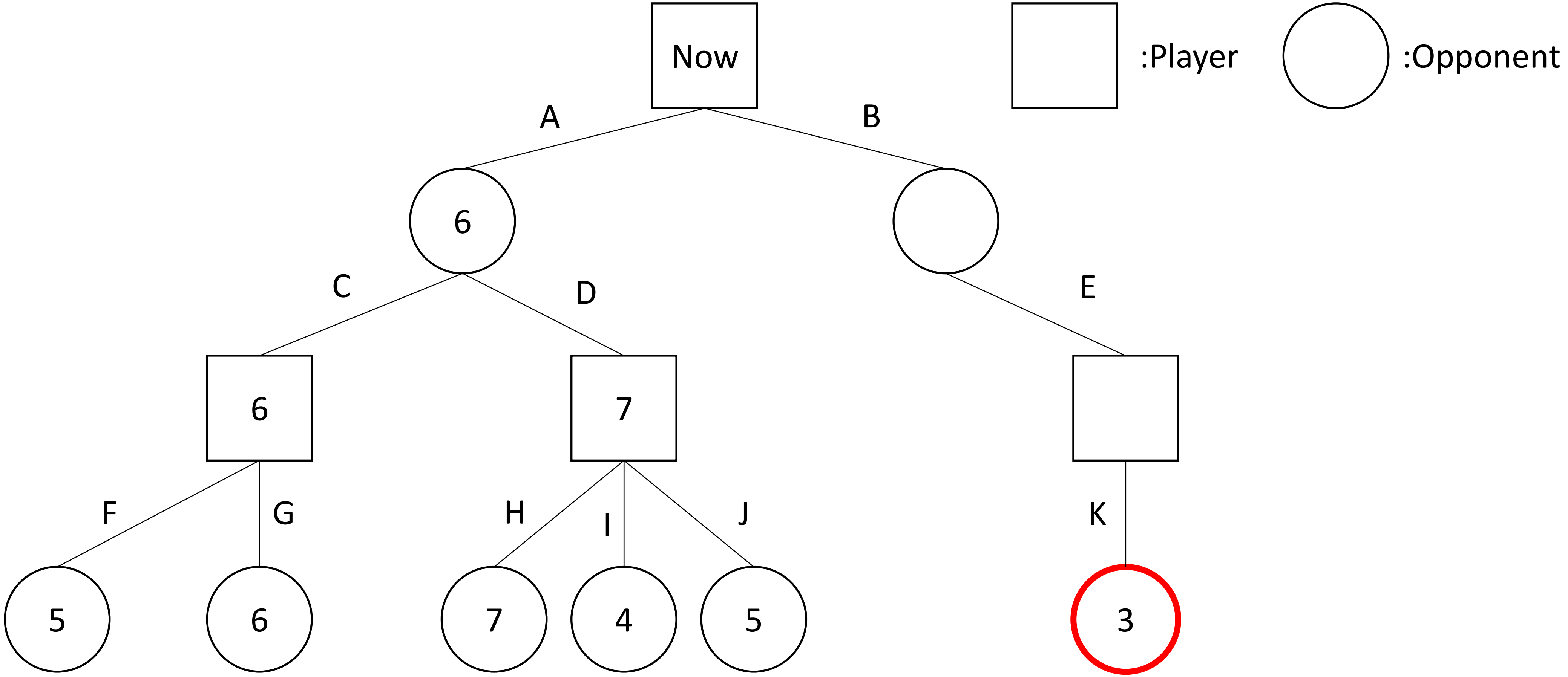
Player picks the largest score



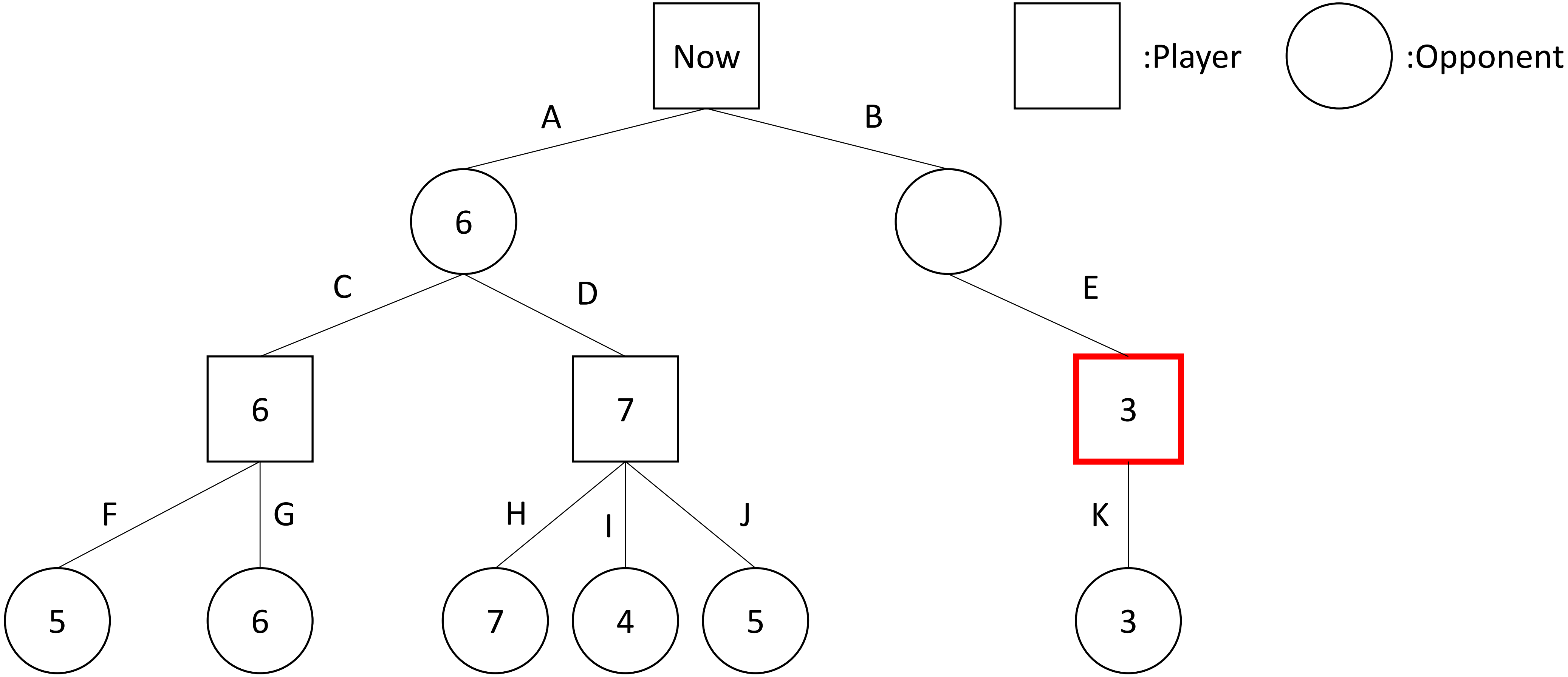
Opponent picks the smallest score



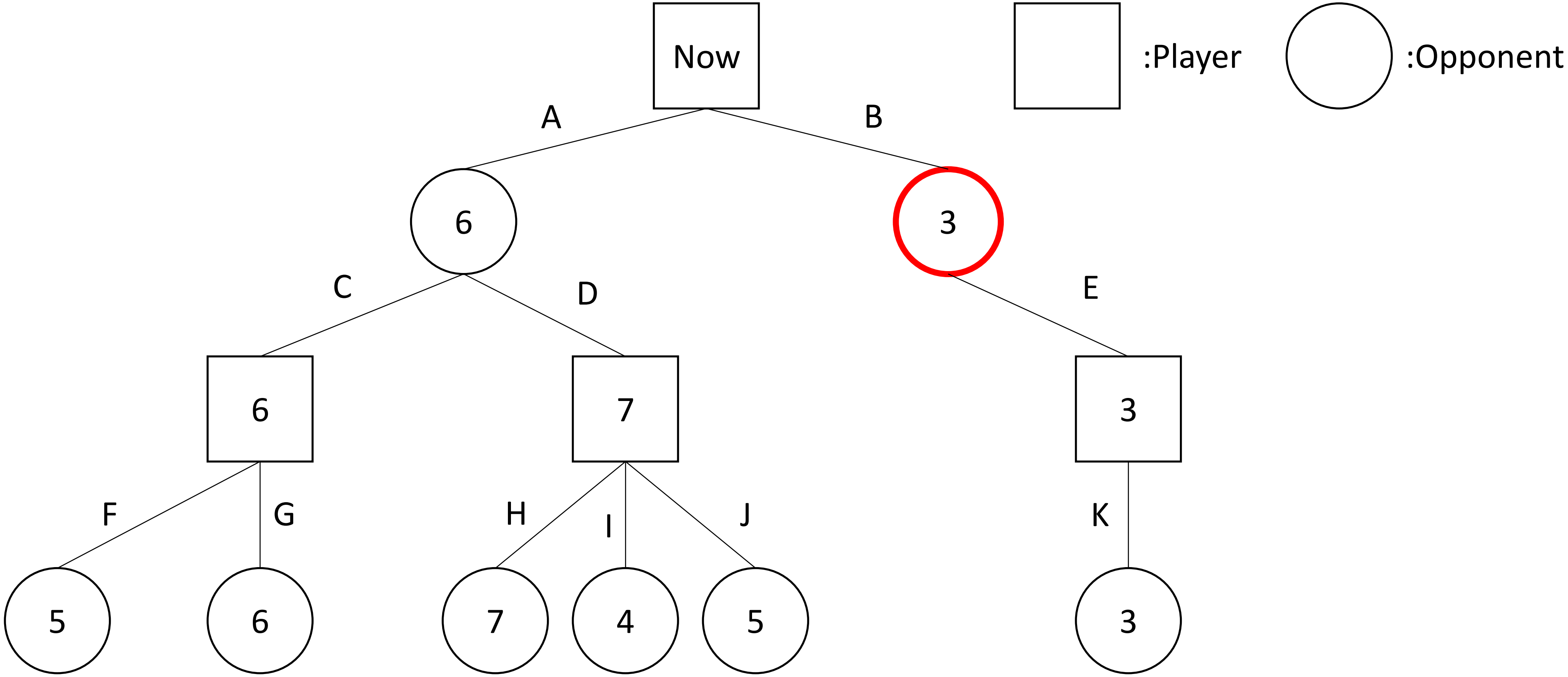
Evaluate score at leaves



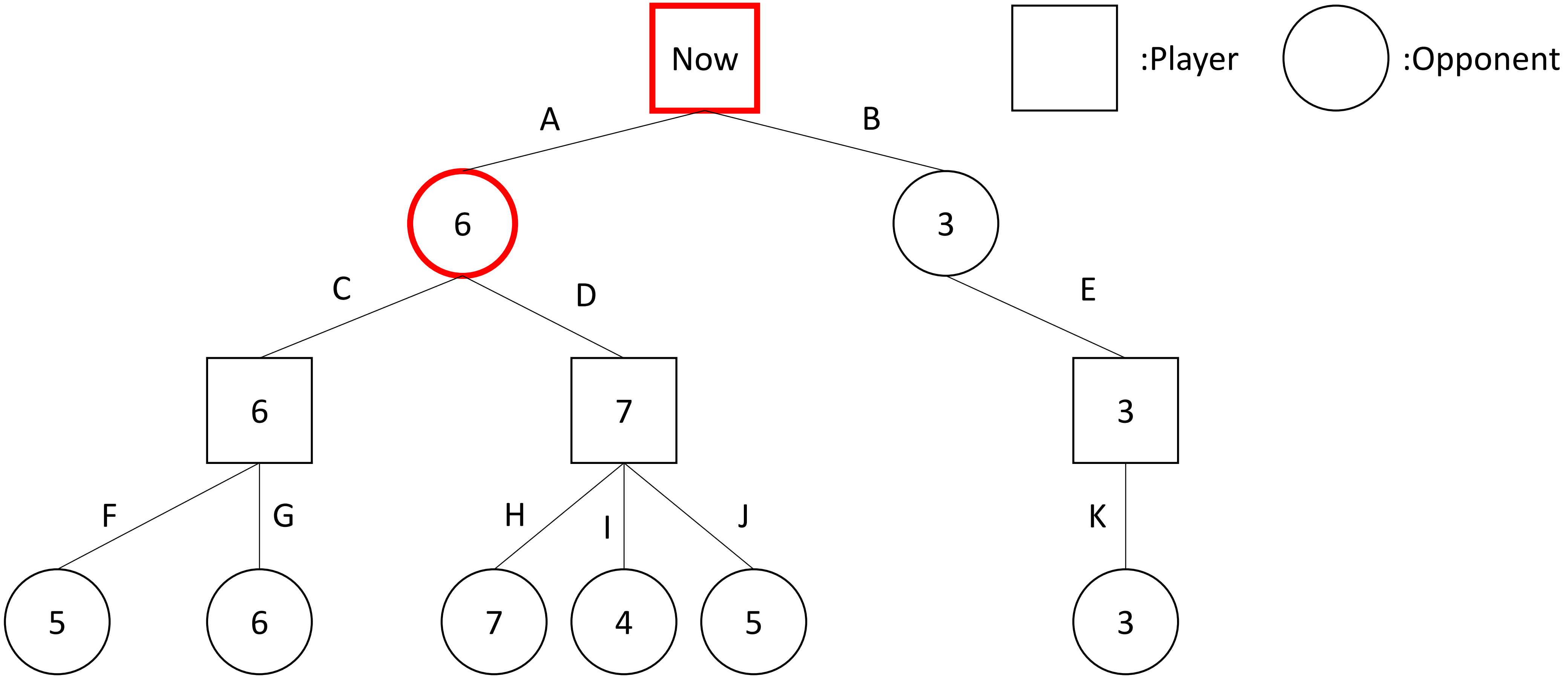
Player picks the largest score



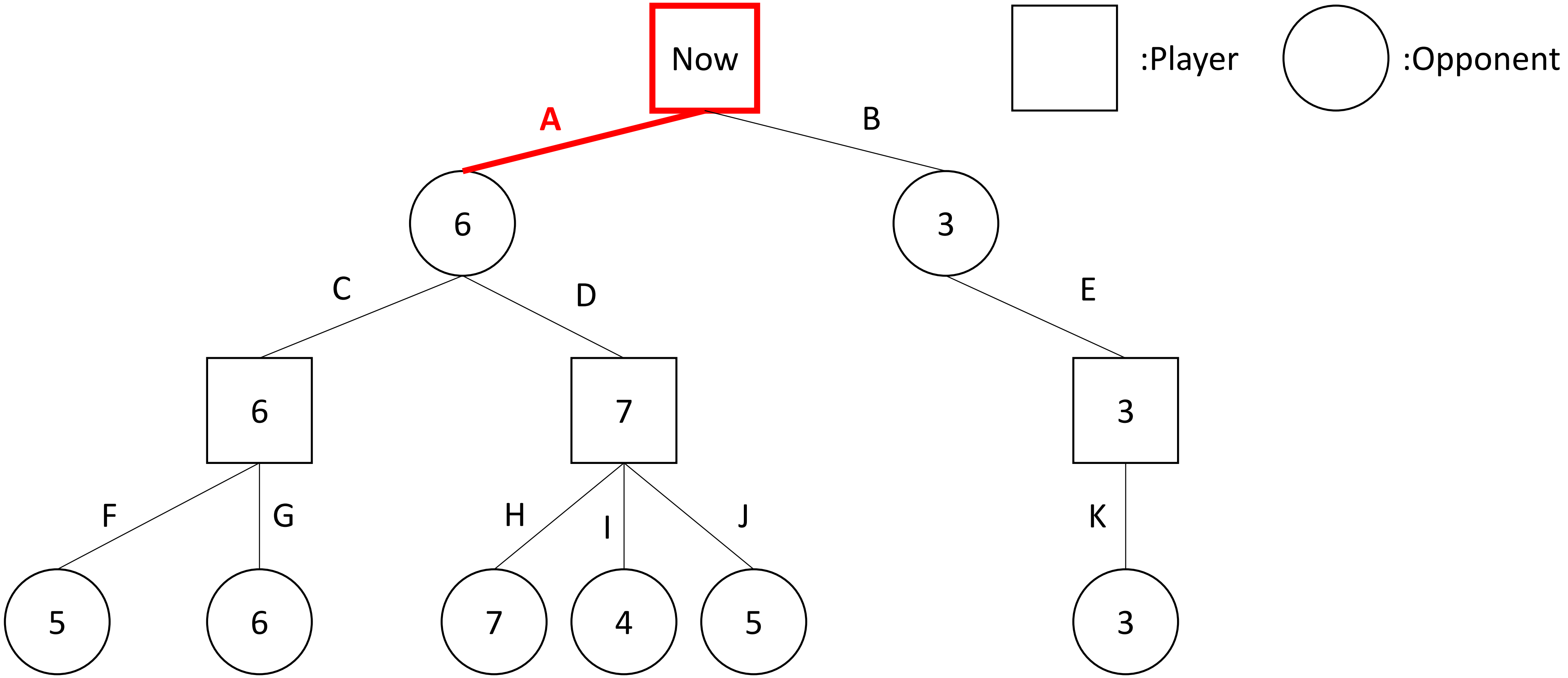
Opponent picks the smallest score



Move A has the largest score



Player picks move A to be the next move



Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

Alpha-Beta Pruning

- By Minimax, we can simulate our opponent’s moves and pick a move with minimum risk and maximum value
- Looking forward for more steps may improve the policy
- However, the size of the search tree may drastically increase with the increase of search depth

Alpha-Beta Pruning

- Since we only have limited time, if we hope to increase search depth, we must optimize the search process
- There are many branches in the minimax process which is not related to the result
- We can try to “prune” these branches to improve efficiency
- The Alpha-Beta Pruning is the improved version of Minimax method which eliminates some unnecessary branches

Alpha-Beta Pruning Pseudocode

```
function alphabeta(node, depth, α, β, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, alphabeta(child, depth − 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cutoff *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, alphabeta(child, depth − 1, α, β, TRUE))
            β := min(β, value)
            if β ≤ α then
                break (* α cutoff *)
        return value
```

Source: https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

Alpha-Beta Pruning

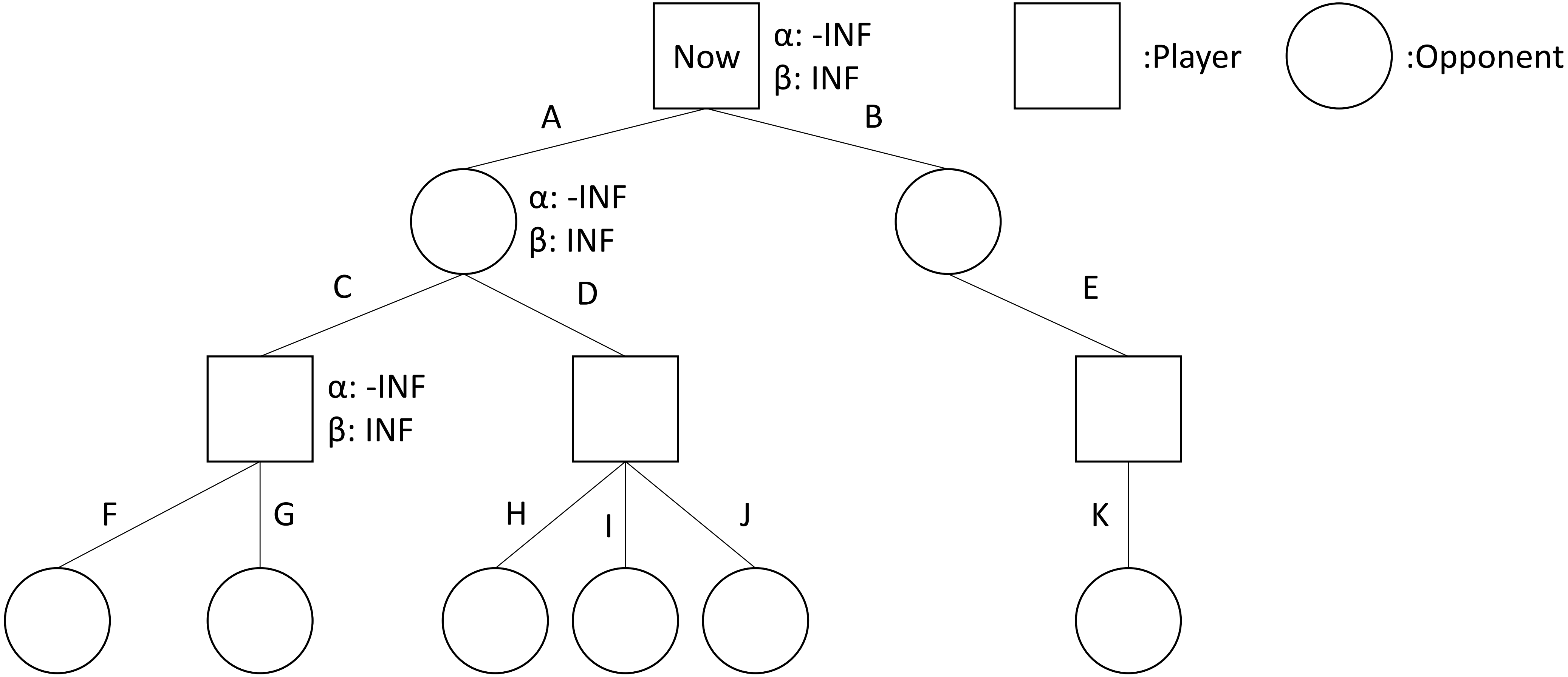
- Alpha: the maximum score that the player is assured of in the current search process

- Beta: the minimum score that the opponent is assured of in the current search process

Alpha-Beta Pruning

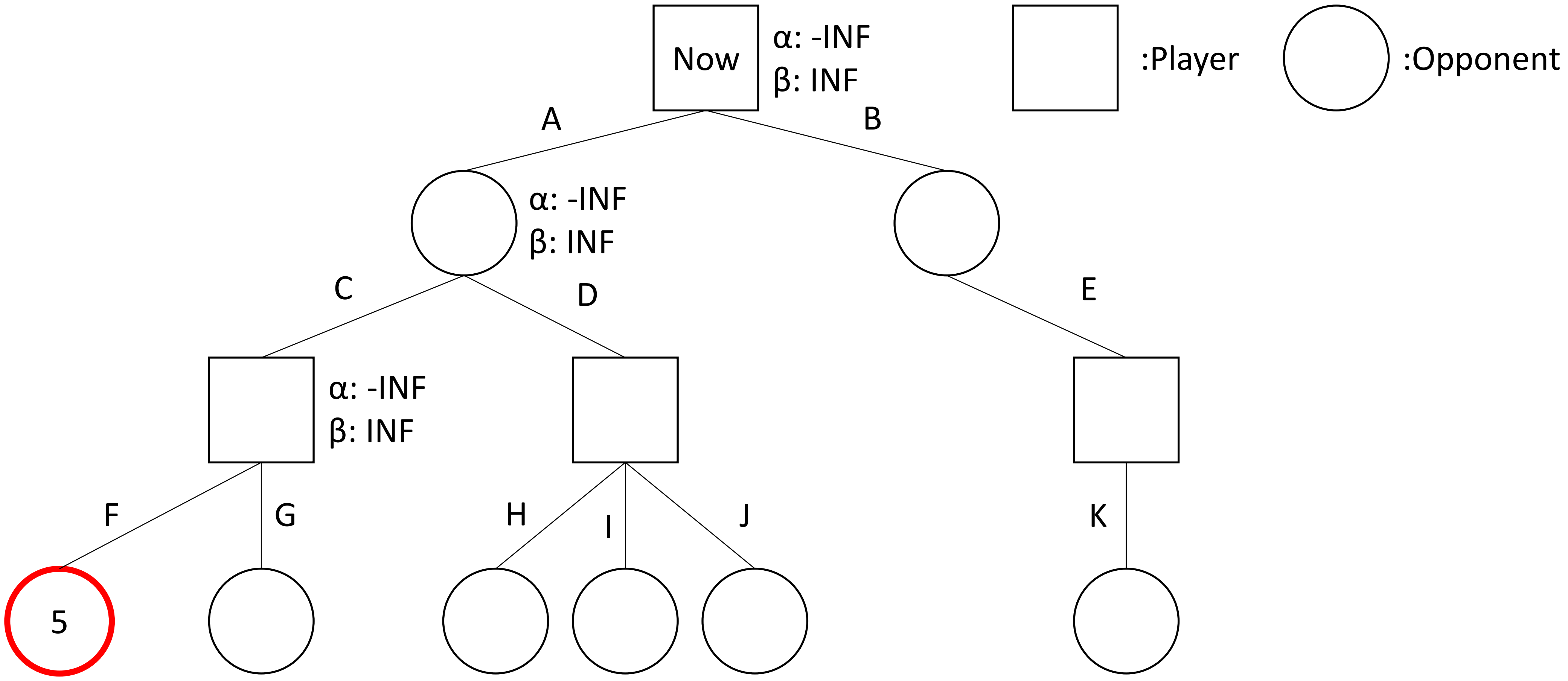
- If alpha >= beta on a player node, we can stop to search on this branch
- In this situation, the player will return a value >= beta on this branch
- However, the opponent already has a better choice (beta)
- Thus, no matter the later discovered value on this branch, the opponent will not pick this branch
- We can "prune" this branch since it will not affect the result
- We can also stop to search if beta <= alpha on an opponent node

Example

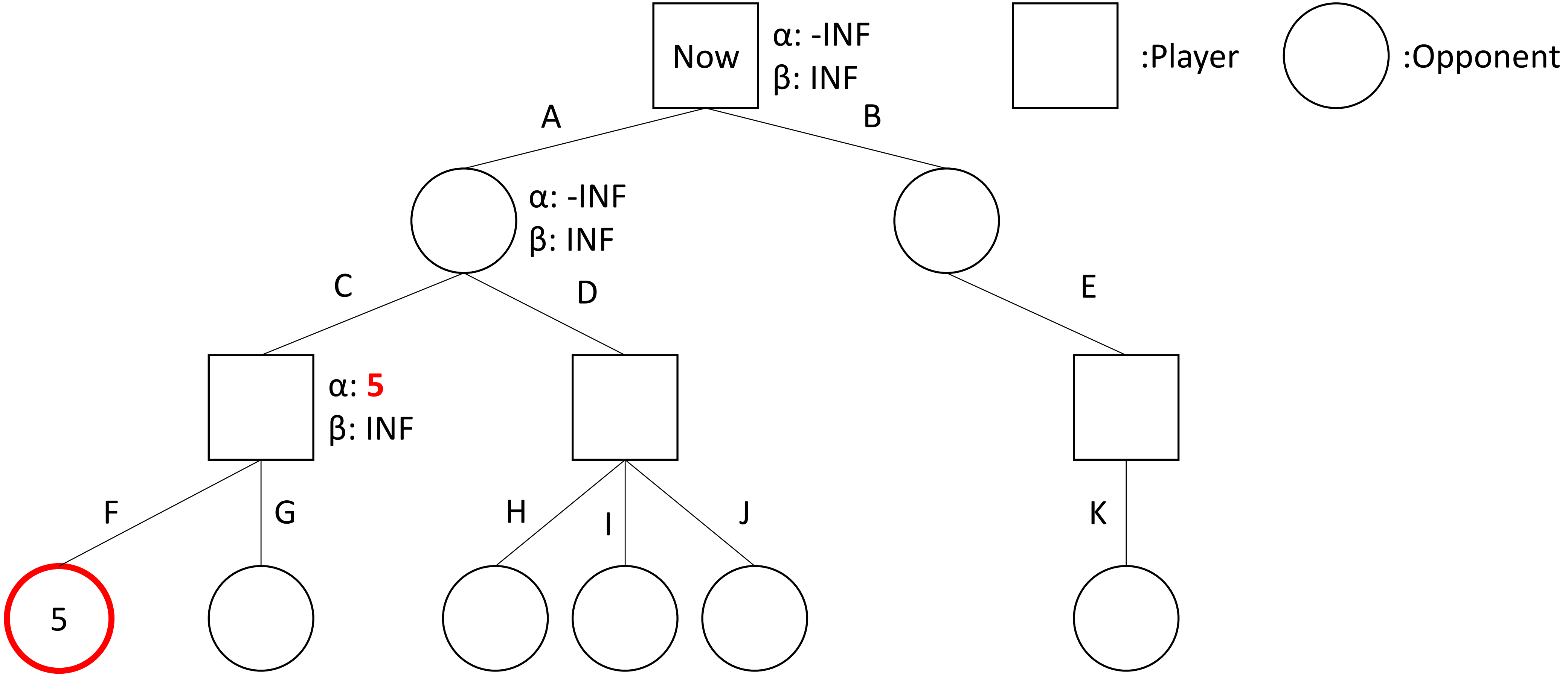


Now
α: -INF
β: INF
:Player
:Opponent
A
B
α: -INF
β: INF
C
D
E
α: -INF
β: INF
F
G
H
I
J
K

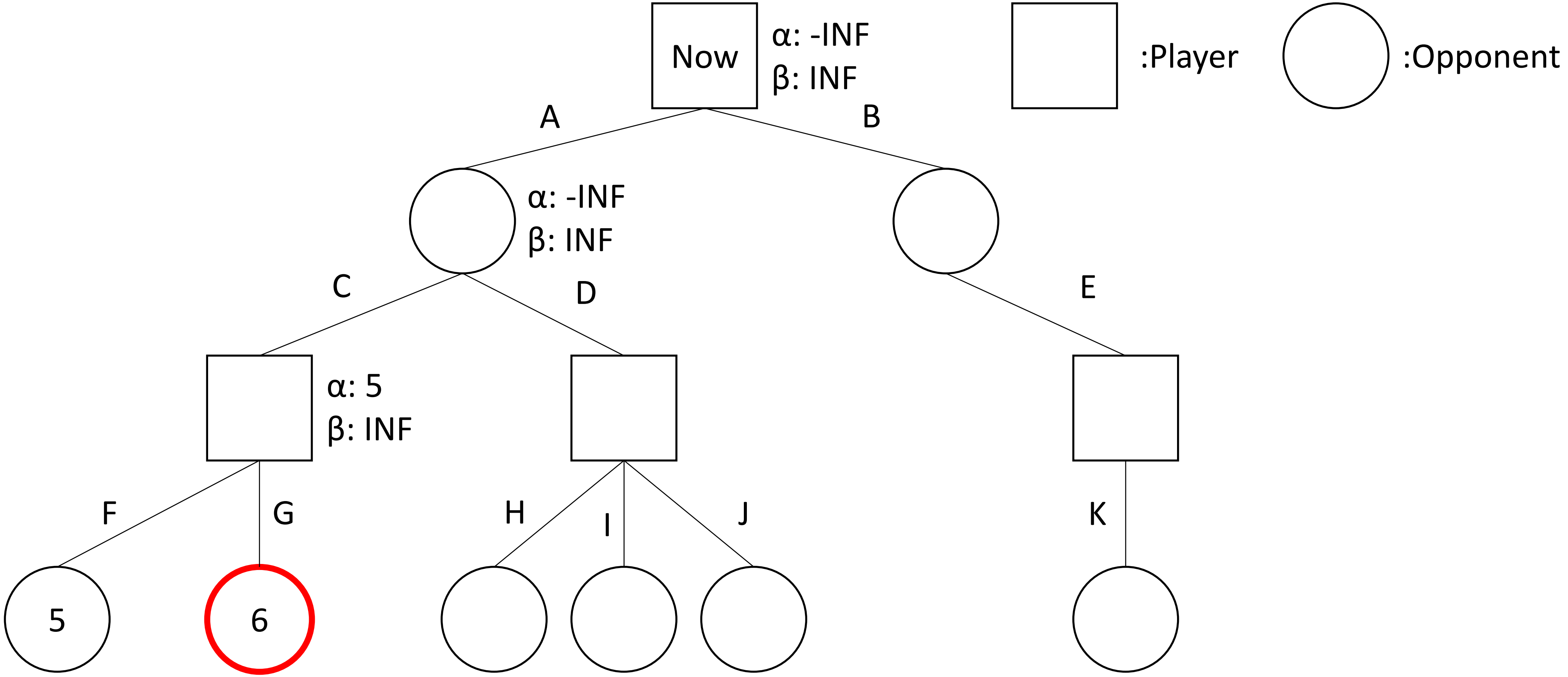
Evaluate score at leaves



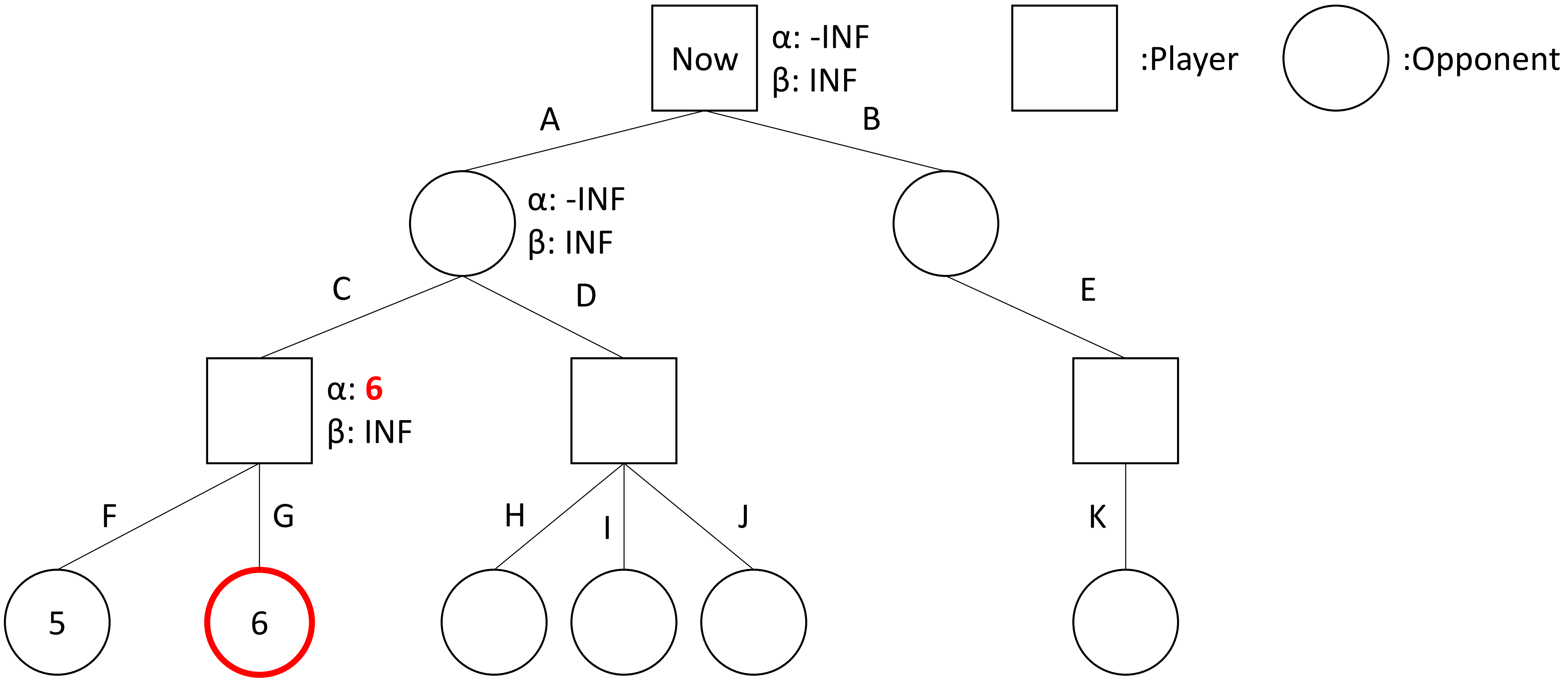
Update alpha



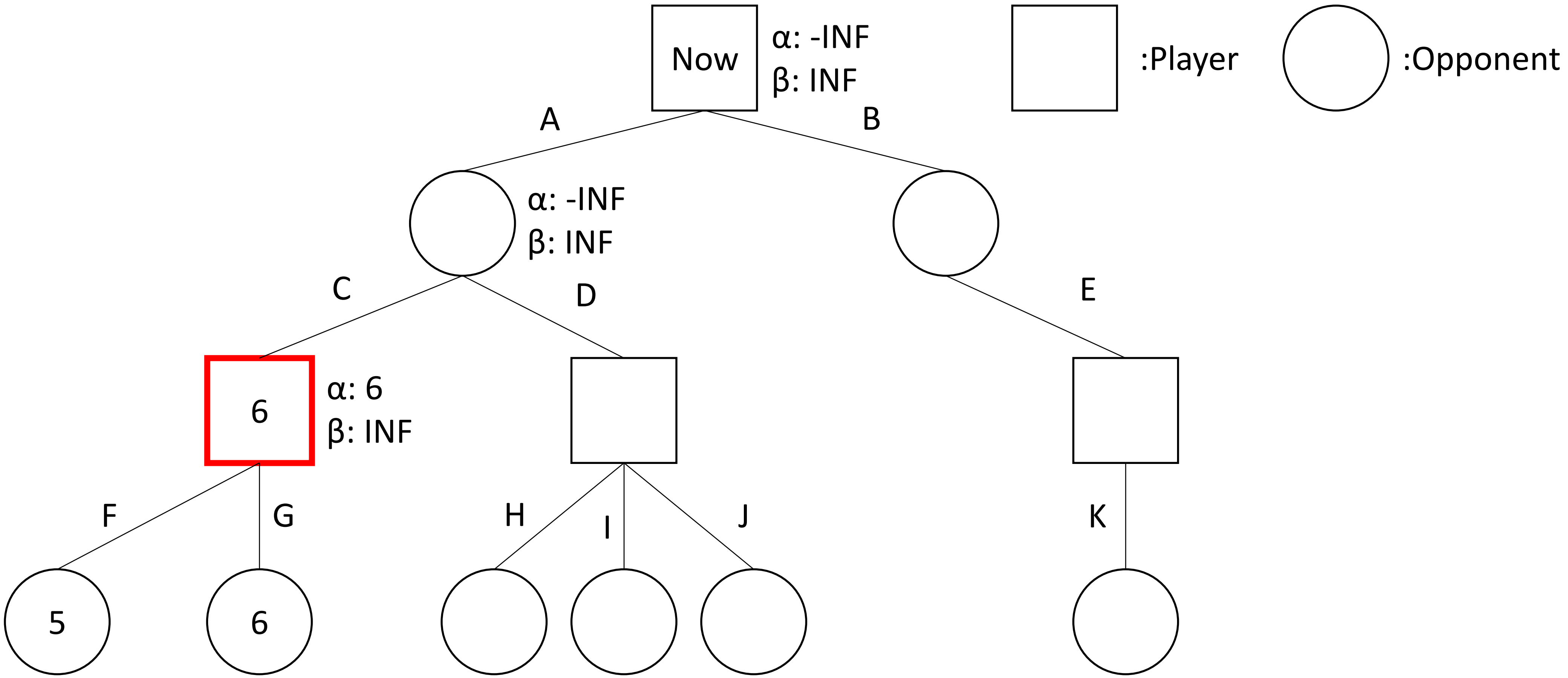
Evaluate score at leaves



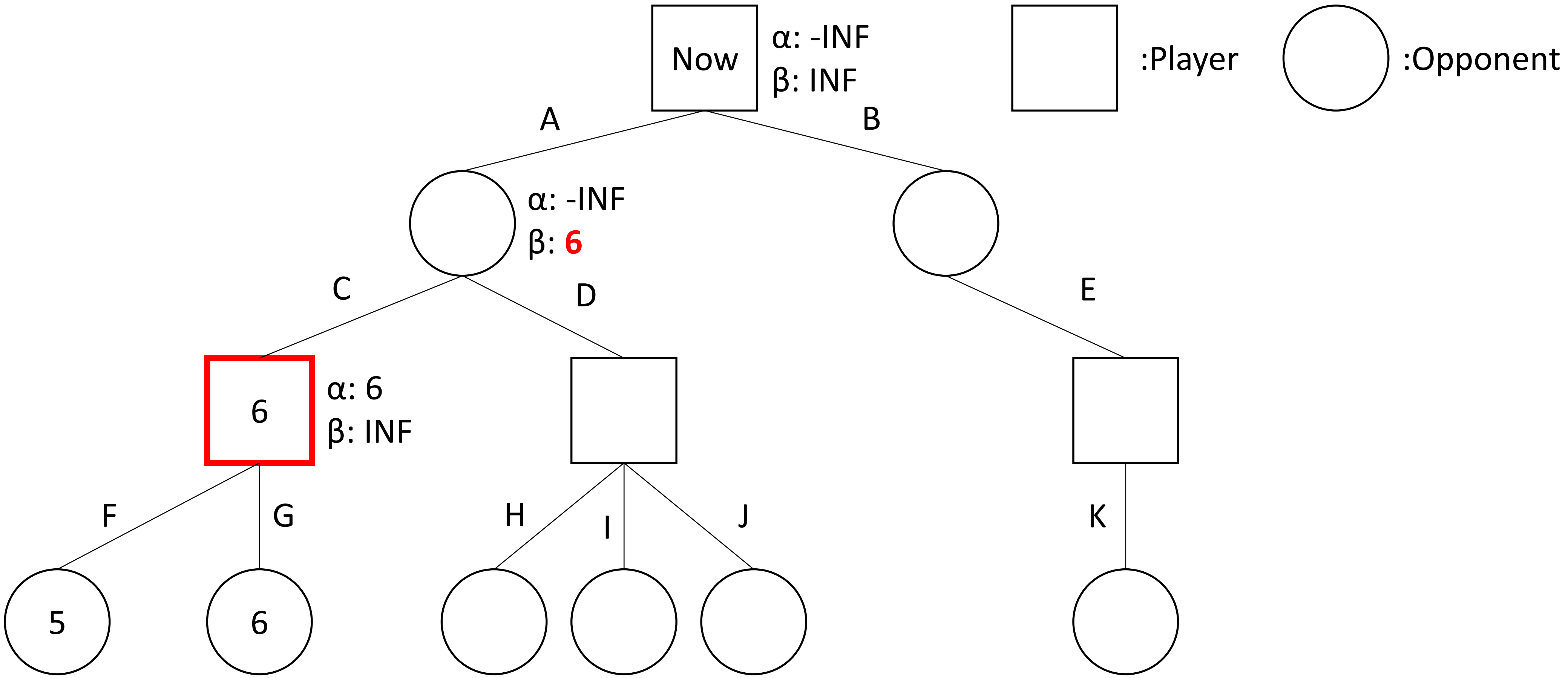
Update alpha



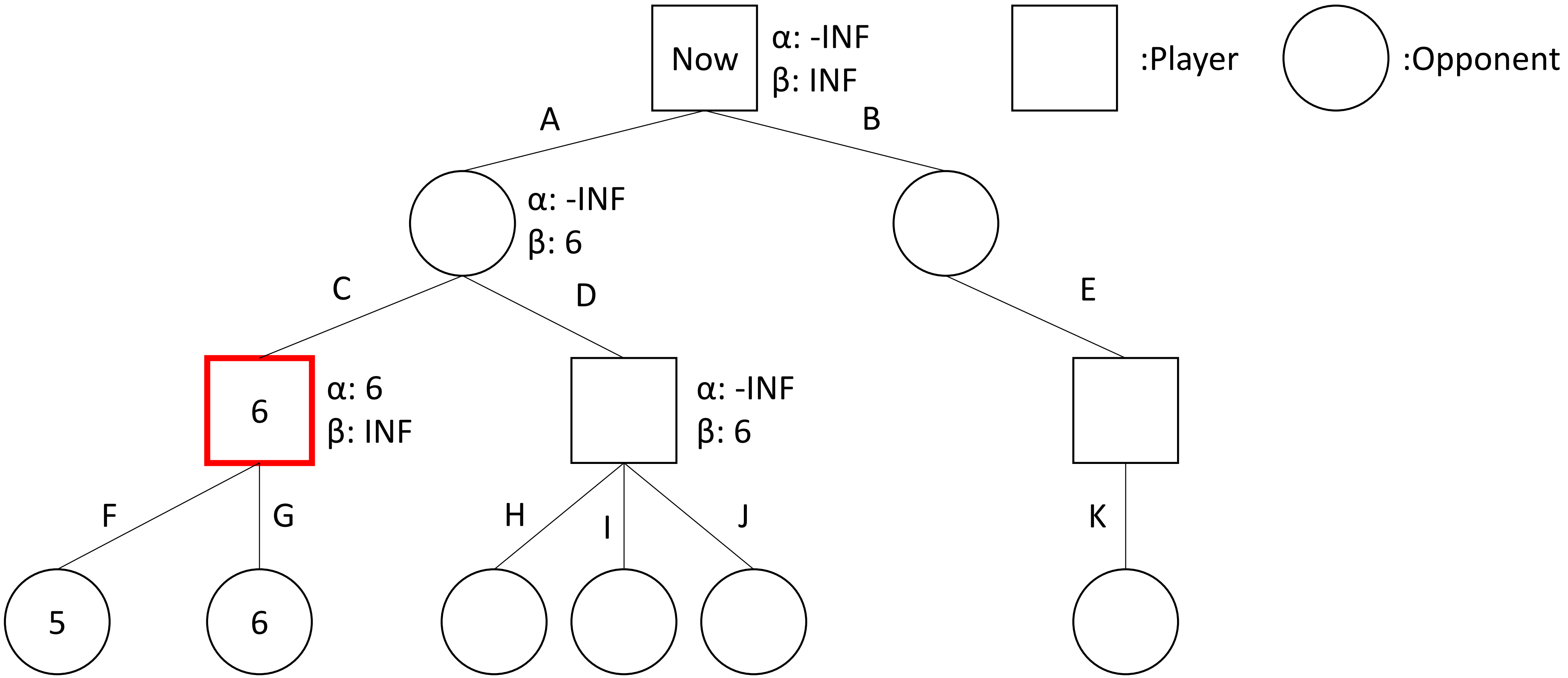
Player picks the largest score



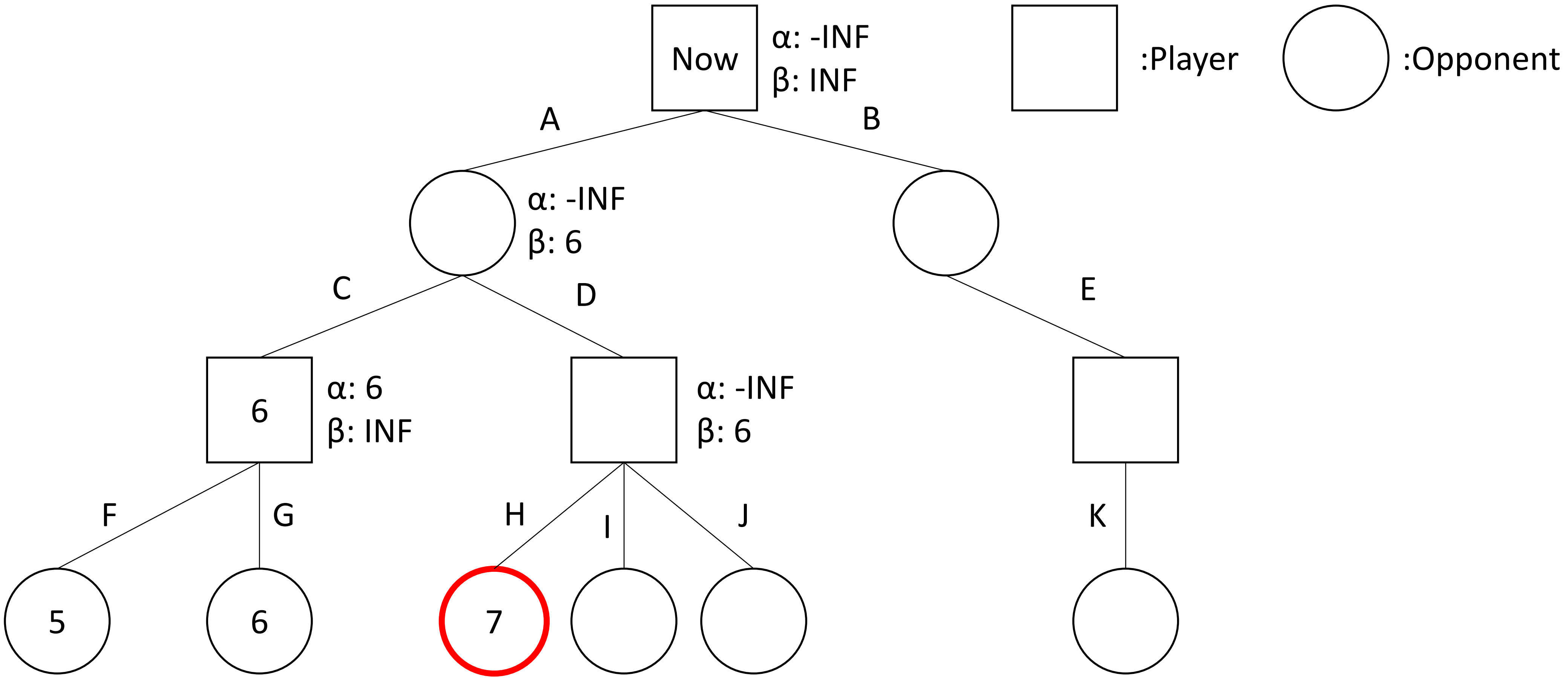
Update beta


Now
α: -INF
β: INF
:Player
:Opponent
A
B
α: -INF
β: 6
C
D
E
6
α: 6
β: INF
F
G
H
I
J
K
5
6

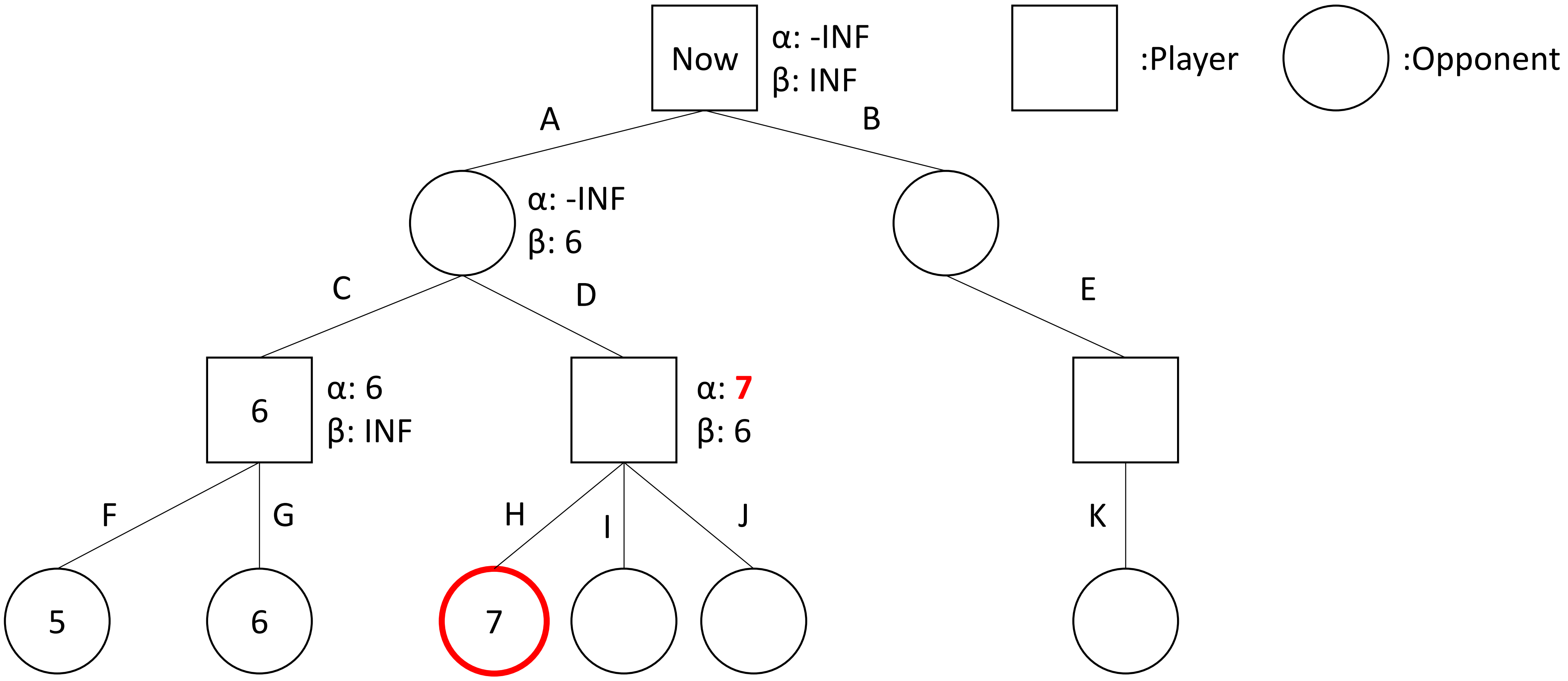
Propagate alpha and beta values



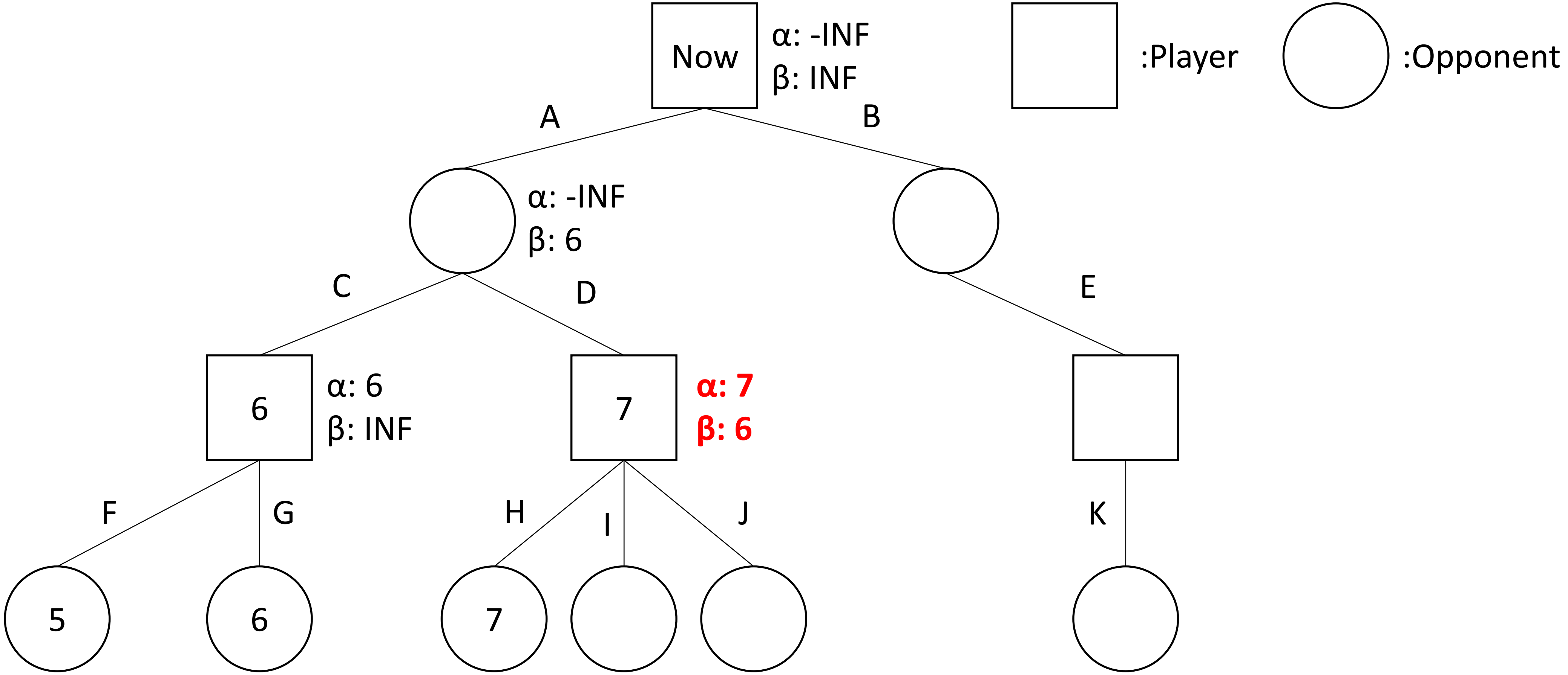
Evaluate score at leaves



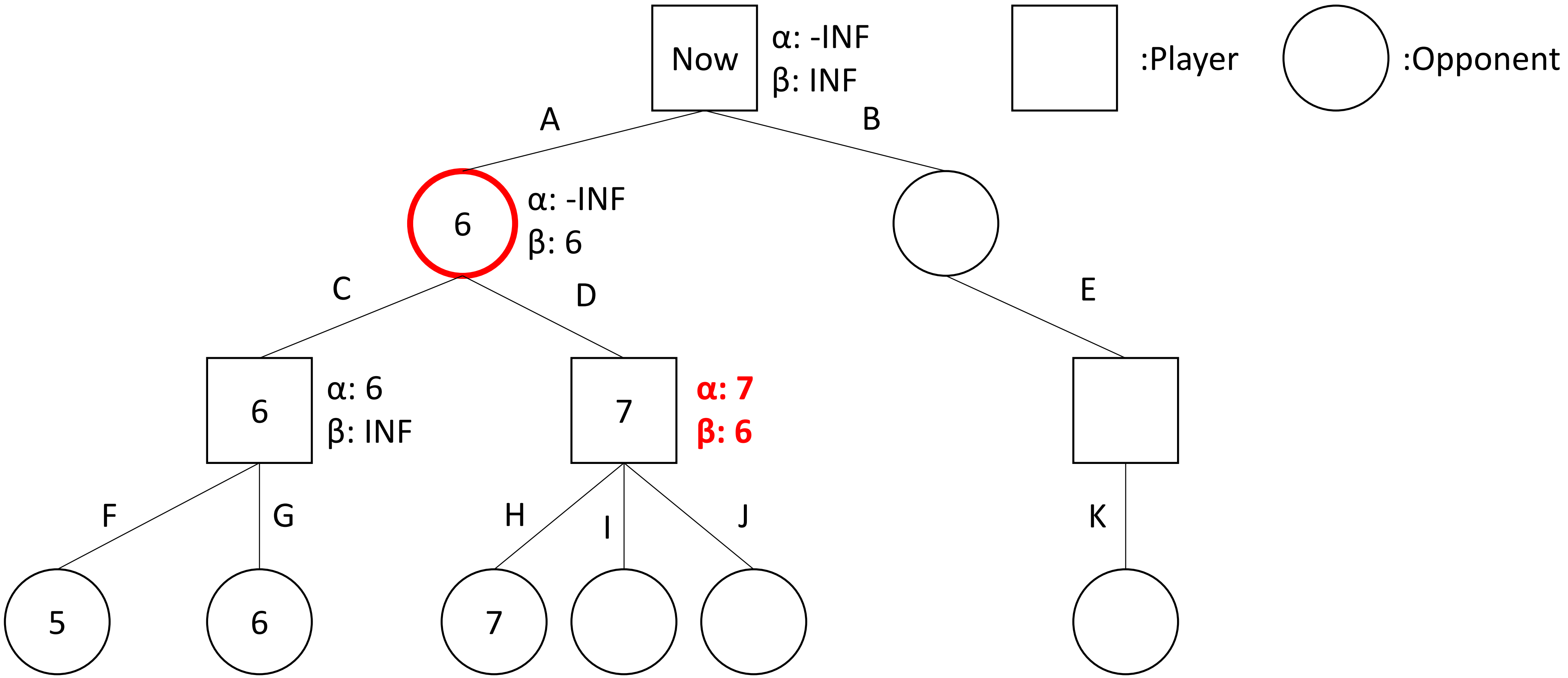
Update alpha



Alpha >= beta in a player node, stop searching



Opponent picks the smallest score



Alpha-Beta Pruning

- In the example above, we use the same search tree as Minimax
- By pruning, we eliminate branches I and J
- However, we still get the same result on branch A
- Alpha-Beta Pruning can effectively speed up the process while maintaining the same result

Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

How To Design Your AI

- The game runner (main.cpp) executes the AIs of the player and the opponent in turns and communicates with them by files
- Your game AI should read the board status from the file “state”
- Your game AI should output your move to the file “action”

State file

- The state file consists of two parts:
- Current player (1 or 2)
- Board (15 x 15 matrix)

```
2
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 2 0 0 0 0 0
0 0 0 1 2 2 2 2 1 0 0 0 0 0 0
0 0 0 0 0 0 2 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Action file

- Your AI should output the next move to the “action” file
- You can keep output moves in the time limit
- Only the last complete output will be considered
  - In the case on the right, (3, 5) will be accepted by the game runner
- You lose if you outputs an invalid move

```
4 2
7 1
3 5
3 5
2 2
2 4
2 4
3 5
3 5
```

How To Design Your AI

- You can refer to the “player_random.cpp” in the “src” folder of the package we provided
- Design your state value function to evaluate the board
- Implement the Alpha-Beta Pruning method and use your value function in the search process
- Run Alpha-Beta Pruning and decide which move to output

Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

Package Directory Structure

- Mini Project 3
  - baseline
    - windows
      - baseline<i>.exe, i := {1, 2}
    - linux
      - baseline<i>, i := {1, 2}
    - mac
      - baseline<i>, i := {1, 2}
  - src
    - main.cpp
    - makefile
    - player_random.cpp
  - documents
    - Mini Project 3 Spec.pdf
    - Mini Project 3 Introduction.pptx
    - Mini Project 3 Environment Settings.pptx
    - Mini Project 3 WSL Installation Tutorial New.pptx

Baseline

- The baseline folder contains executables of the baselines
- If you use Windows, use the baselines in the windows folder
- If you use Linux, use the baselines in the linux folder
- If you use Mac, use the baselines in the mac folder
- We have three hidden baselines and two open baselines to test the performance of your program
- Test your program against the open baselines and try to defeat them

Baseline

- baseline0 => pure random
- baseline1 => minimax with basic value function
- baseline2 => deeper minimax with basic value function
- baseline3 => alpha beta pruning with basic value function
- Baseline4 => deeper alpha beta pruning with basic value function

Src

- The src folder contains the main program, a makefile and some provided example programs
- The main.cpp is the game runner, do not modify it
  - However, you can refer to it when implementing game simulation of minimax
- The makefile helps you compile your code
  - You can also compile by hand, make sure you set the flags -Wall and –Wextra
- player_random.cpp => the pure random AI
  - You can refer to this one and write your own code

How To Compile Your Code

- To compile all programs:
- Type make when your working directory is ‘src’

- To compile <program_name>.cpp:
- Type make <program_name> when your working directory is ‘src’

- To clean the executables:
- Type make clean when your working directory is ‘src’

How To Test Your Code

- If you use Windows, type the command below in cmd:
- main.exe <AI1>.exe <AI2>.exe

- If you use Linux / Mac, type the command below in terminal:
- ./main ./<AI1> ./<AI2>

Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

Requirements - Code

- Design and implement an AI that can play the boardgame Gomoku
- Your AI should read the board from file and write next move to file
- Design a value function to evaluate the score of the board
- Enhance the policy of your AI with Alpha-Beta Pruning

Requirements - Code

- If you are not satisfied by the Alpha-Beta Pruning algorithm, you can try some more advanced methods. However, make sure you can explain how Minimax and Alpha-Beta Pruning works during demo

- If you cannot complete the Alpha-Beta Pruning algorithm, implementing the basic Minimax algorithm also gives you some score

Requirements - Code

- You will lose immediately if your program outputs an invalid move
- Time limit for each move is 10 seconds, and the memory limit is 4GB
- You can keep output moves in the limited time. Only the last successful output move is used by the game runner
- Please refer to the spec for more detailed rules

Requirements - Submission

- Please use C++ and write your program in a single file
- Your program should be named as <student_id>_project3.cpp
- Your program will be compiled in a GNU / Linux environment by:
- g++ -O2 -std=c++14 -Wall -Wextra <student_id>_project3.cpp
- Please make sure your program can be compiled by the command above with no error

Requirements - Report and Demo

- You should write a report to elaborate on how you design your AI
- The report is not directly graded, but is your only available reference through the TA demo (You cannot refer to your code in demo)
- You must attend the demo and answer the questions from TA
- The demo date and method will be announced soon

Outline

1. Introduction
2. Gomoku
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading

Grading

- The project accounts for 9 points of your total grade
- Beat every baseline => +5 points (1 point for each baseline) (Win First Hand and Second Hand)
- Implement Tree search (Minimax) => +2 points
- Design of your state value function => +1 point
- Implement Alpha-Beta Pruning => +1 point

Grading Bonus

- (Bonus) Uses version control software => +1 point
- Include a screenshot with more than 3 commits in your report

- (Bonus) Class ranking => At most +2 points
- You can attend the class ranking if you beat all baselines
- Your AI will play against other AIs of your classmates and gain bonus score according to your ranking

Happy Coding!