



Mini Project 3

Mini Chess AI

Due: 6/20

Demo: 6/21 online

Outline

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

Outline

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

Introduction

- ♦ Design and implement an AI which can play MiniChess
- ♦ 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 a tree search algorithm

Outline

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

Chess



Deep Blue (chess computer) - Wikipedia

Creator: Jim Gerdner - <http://www.ibm.com>

Want to know where this information comes from? [Learn more](#).
Images may be subject to copyright. [Learn More](#)

Mini Chess

- Has lot of variance.
- We use “**MinitChess**” in this project



Basic rules

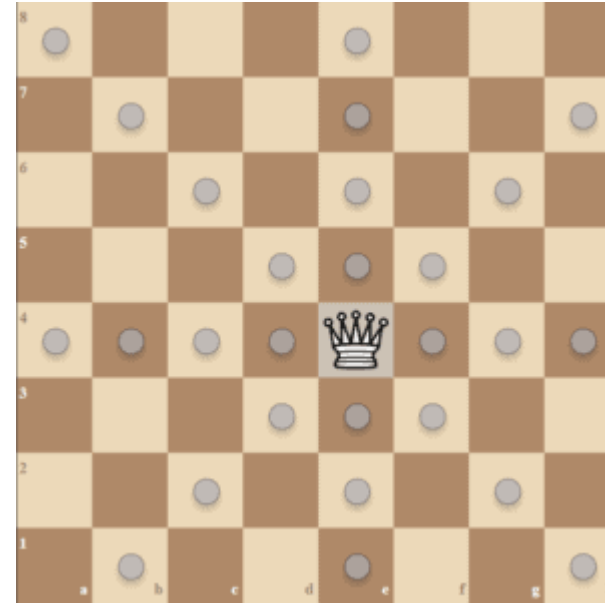
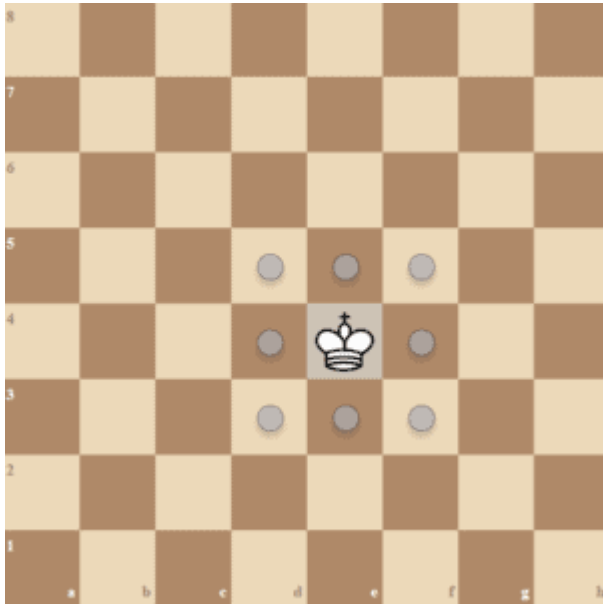
- ♦ We have 2 players: white and black. White plays first.
- ♦ If the target place of your piece has the opponent's piece, you can take it out (or, catch the piece), You cannot take your own piece.
- ♦ If a player can catch the opponent's king in its turn, it wins.
 - ♦ So a player can only win in he turn or lose in the opponent's turn.
- ♦ If a player makes an illegal move, he loses.
- ♦ Our rule is simplified, any different rules will be marked as red one:
- ♦ There is no castling
- ♦ Pawn can only promote to Queen (detail in next section)

Basic rules

- ♦ If 2 players have almost the same ability, it is very likely to get a draw and fall into an infinite loop. So we add these rules:
 - ♦ If it cost over 50 step(25 turn), we count the piece value.
Queen=20, Bishop=8, Knight=7, Rook=6, Pawn=2.
The player who has a higher piece value after 50 steps, wins.
 - ♦ If both sides have the same value, it is a draw.
 - ♦ The Queen who is promoted by Pawn, counts as queen.

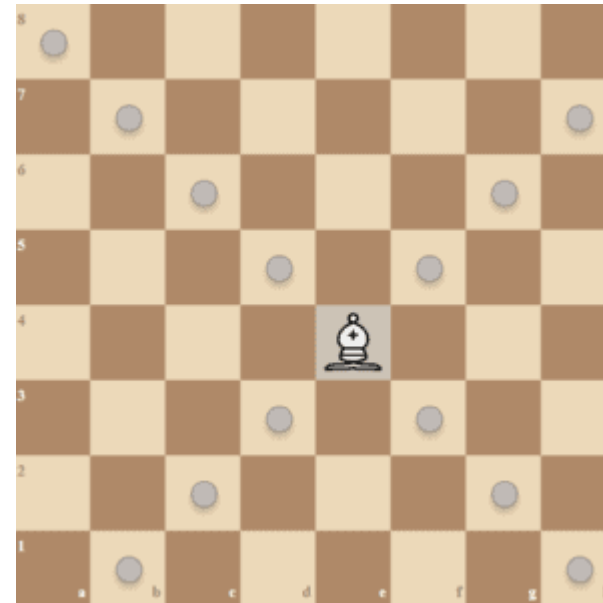
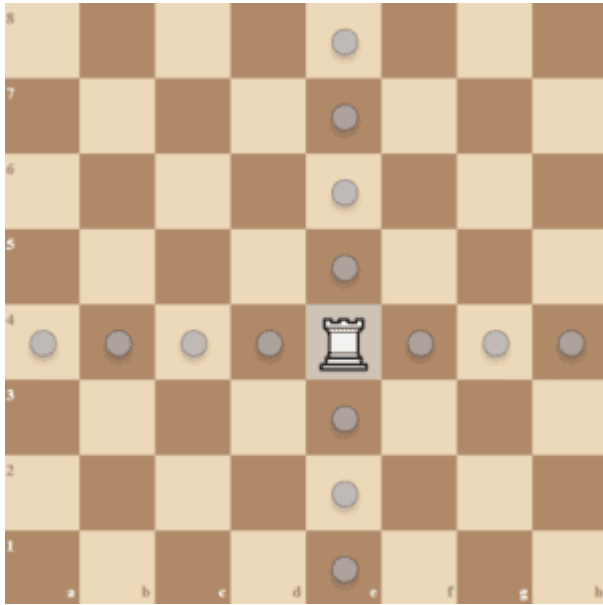
Piece Movement

- ♦ King and Queen



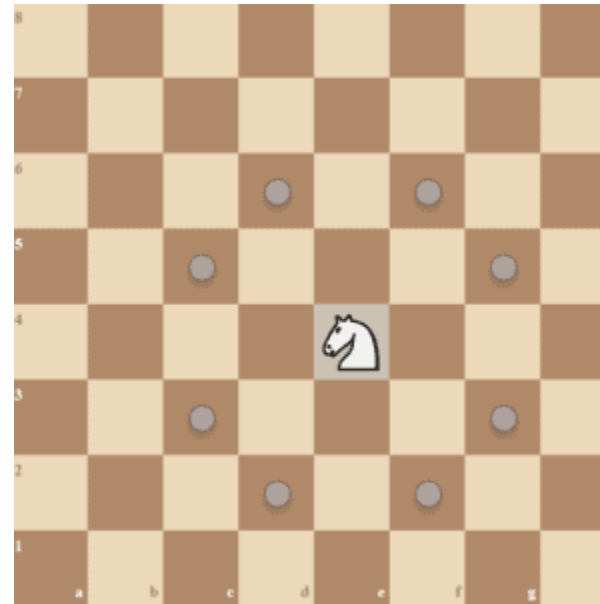
Piece Movement

- ♦ Rook and Bishop



Piece Movement

- ♦ Knight
- ♦ Knight can “jump over pieces.”
 - ♦ It cannot be blocked



Piece Movement

- ♦ Pawn
- ♦ Every turn, the pawn can move forward one step.
 - ♦ Even in the first move of that pawn.
 - ♦ With ↑ this rule, there is no En passant.
- ♦ If the left/right forward is the opponent's piece, you can catch it.
- ♦ When a pawn move to the last row (6 for white, 1 for black), it becomes Queen (promotion).
 - ♦ Promotion and Move to the Last Row will happen simultaneously.



Outline

1. Introduction
2. Chess and Mini Chess
3. **State Value Function**
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading
10. Submission

State Value Function

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

State Value Function

- ♦ State \Rightarrow the board
- ♦ Value \Rightarrow how “good” the board is
- ♦ Function \Rightarrow given a board, output the value

State Value Function

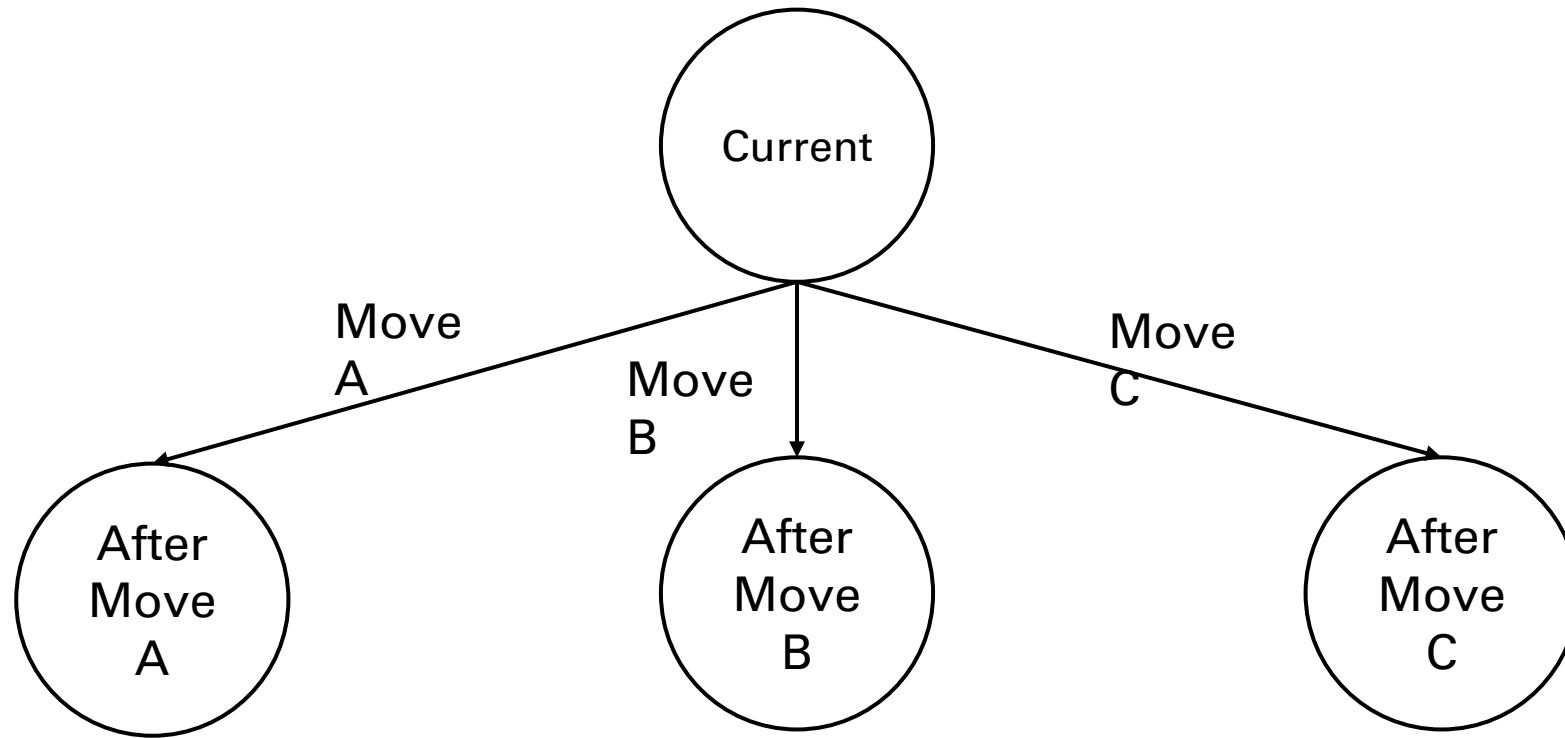
- ♦ Super simple Example:
- ♦ Give every piece a score (king=inf, queen=100, ...)
- ♦ Your pieces – Opponent's pieces = value of the state.
- ♦ Some upgrades:
- ♦ A piece in a different place has a different value.

State Value Function

- ◆ Keywords for a more complicated algorithm:
- ◆ KP (King-Piece), PP (Piece-Piece), KPPT (King-Piece-Piece-Turn)
- ◆ KKPT (King-King-Piece-Turn with King-Piece-Piece)
- ◆ MCTS (Monte Carlos Tree Search)
- ◆ AlphaZero
 - ◆ Leela Chess Zero
 - ◆ DLShogi
- ◆ NNUE (Efficiently Updatable Neural Network)
 - ◆ This is the SOTA for Chess and Shogi
 - ◆ Stockfish (2022 TCEC 1st place, 2022 CCC 1st place)
 - ◆ 水匠 (第3回世界将棋AI電竜戦優勝)

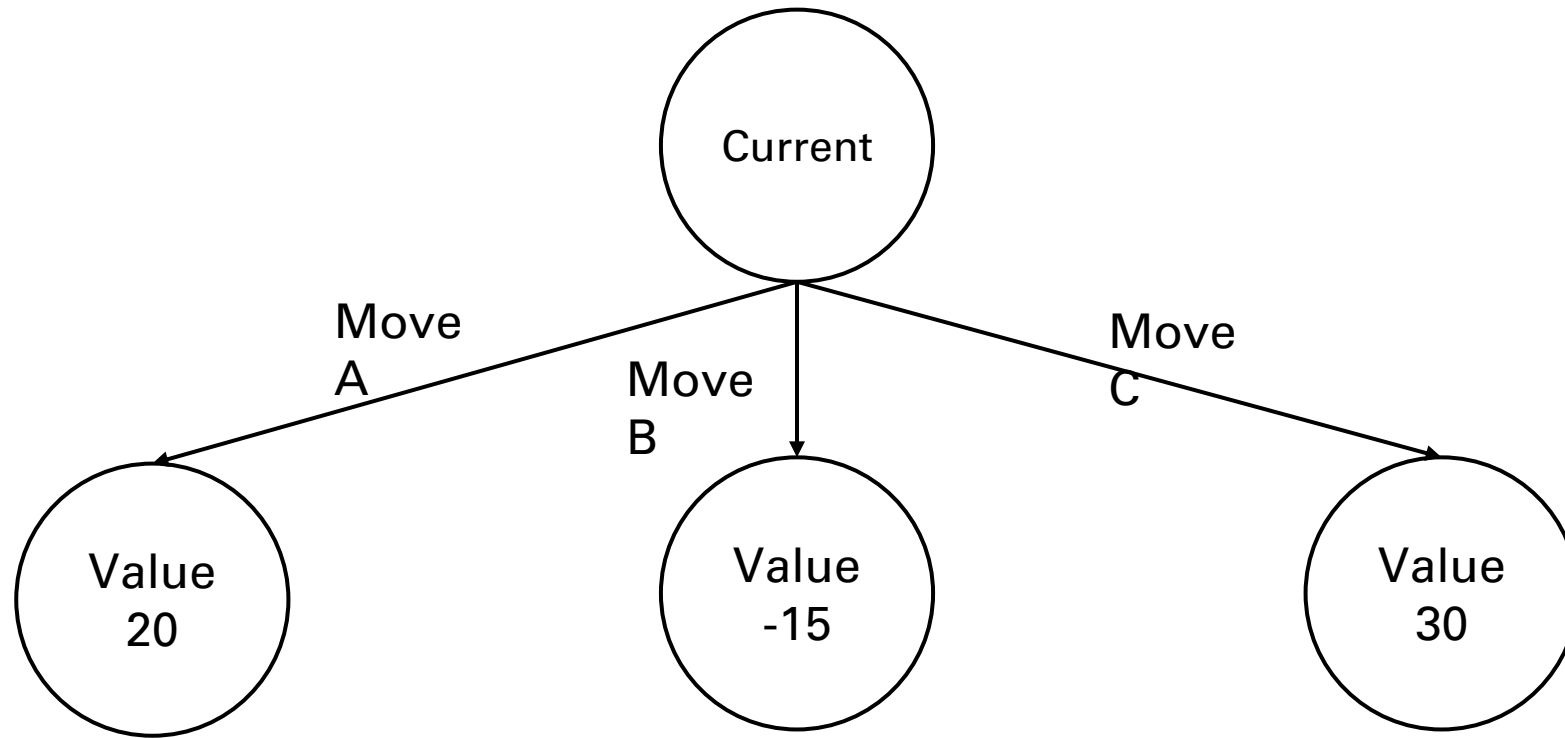
Use value function to pick the next move

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



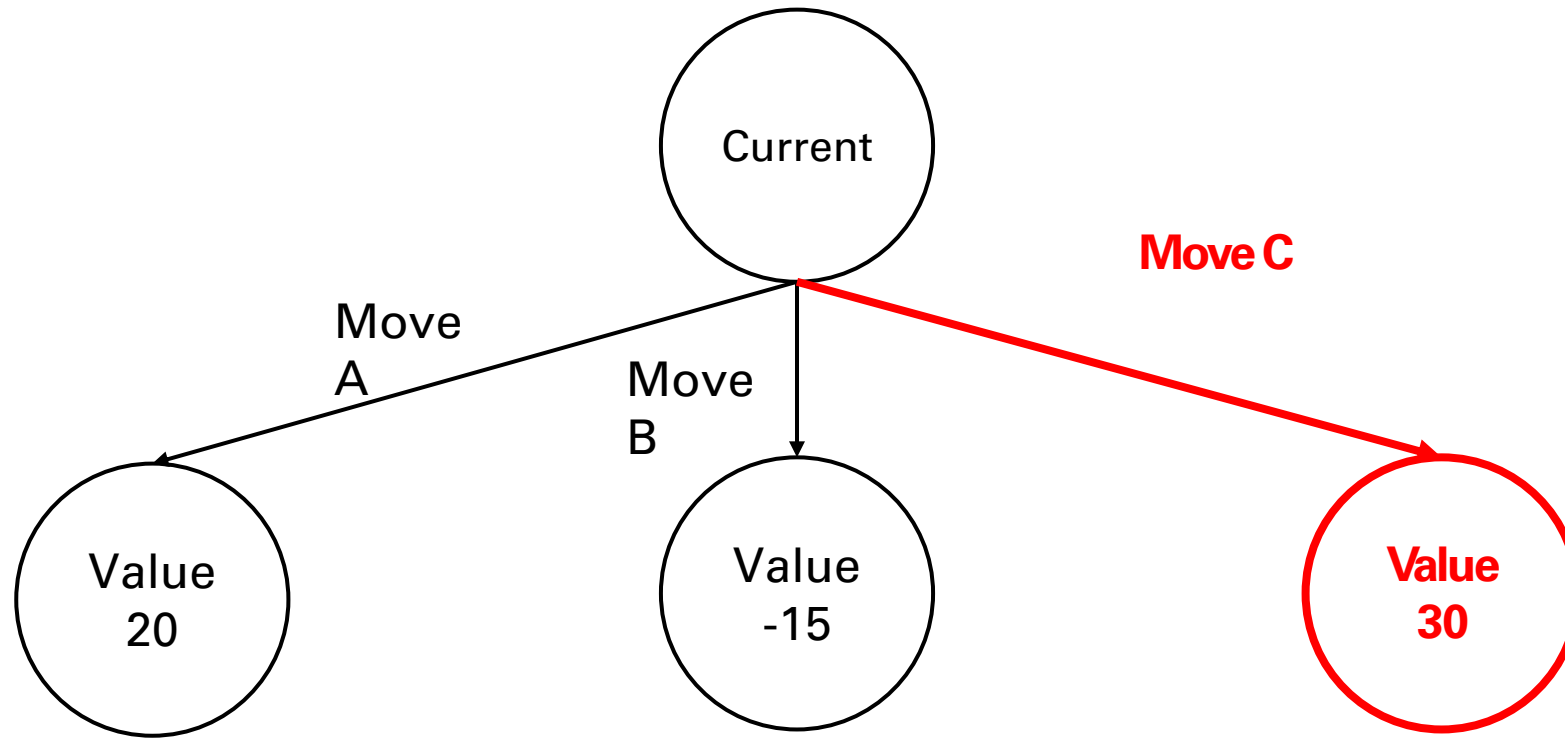
Use value function to pick the next move

- 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



Outline

1. Introduction
2. Chess and Mini Chess
3. State Value Function
4. **Minimax**
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading
10. Submission

Minimax

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

Minimax

- ♦ Player tries his 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, the opponent tends to **give the player the worst board**
- ♦ **The Minimax algorithm is based on this player-opponent interaction**

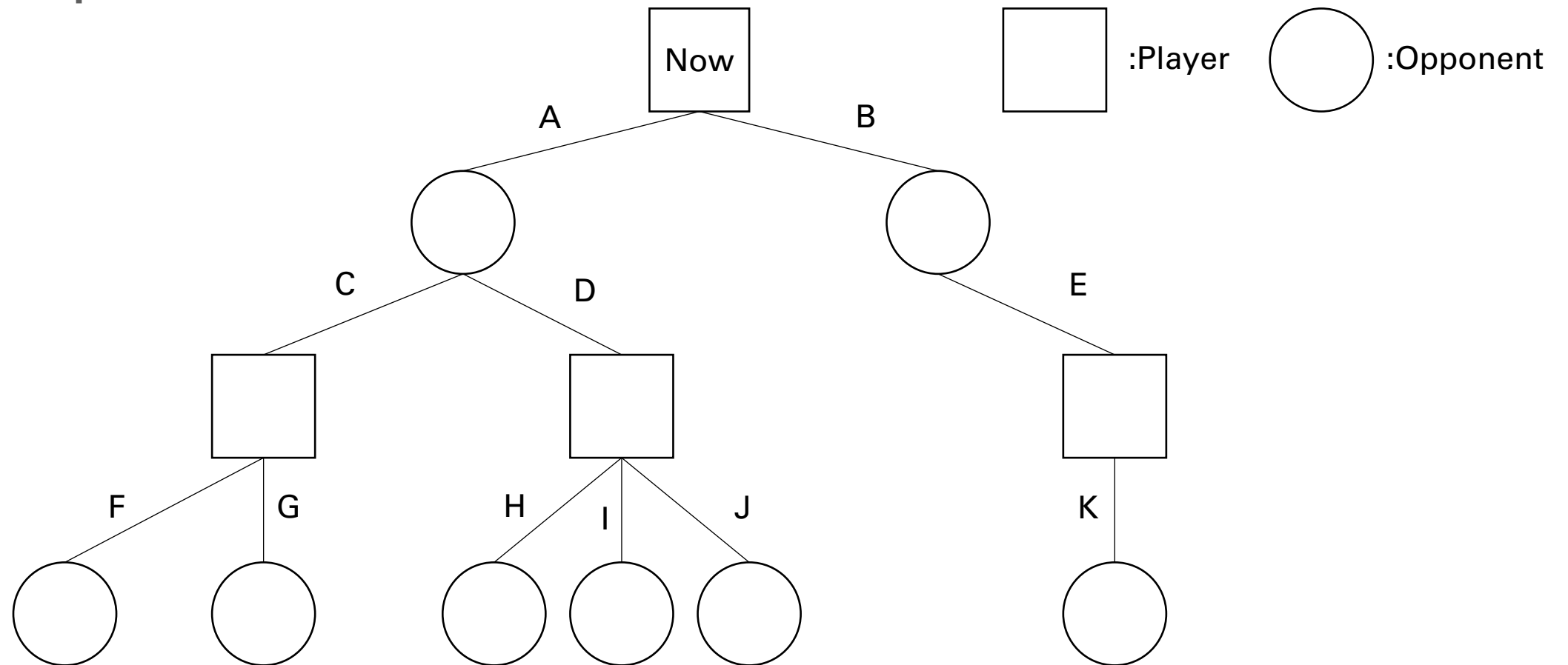
Minimax Pseudocode

```
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 :=  $-\infty$   
        for each child of node do  
            value := max(value, minimax(child, depth - 1, FALSE))  
        return value  
    else (* minimizing player *)  
        value :=  $+\infty$   
        for each child of node do  
            value := min(value, minimax(child, depth - 1, TRUE))  
    return value
```

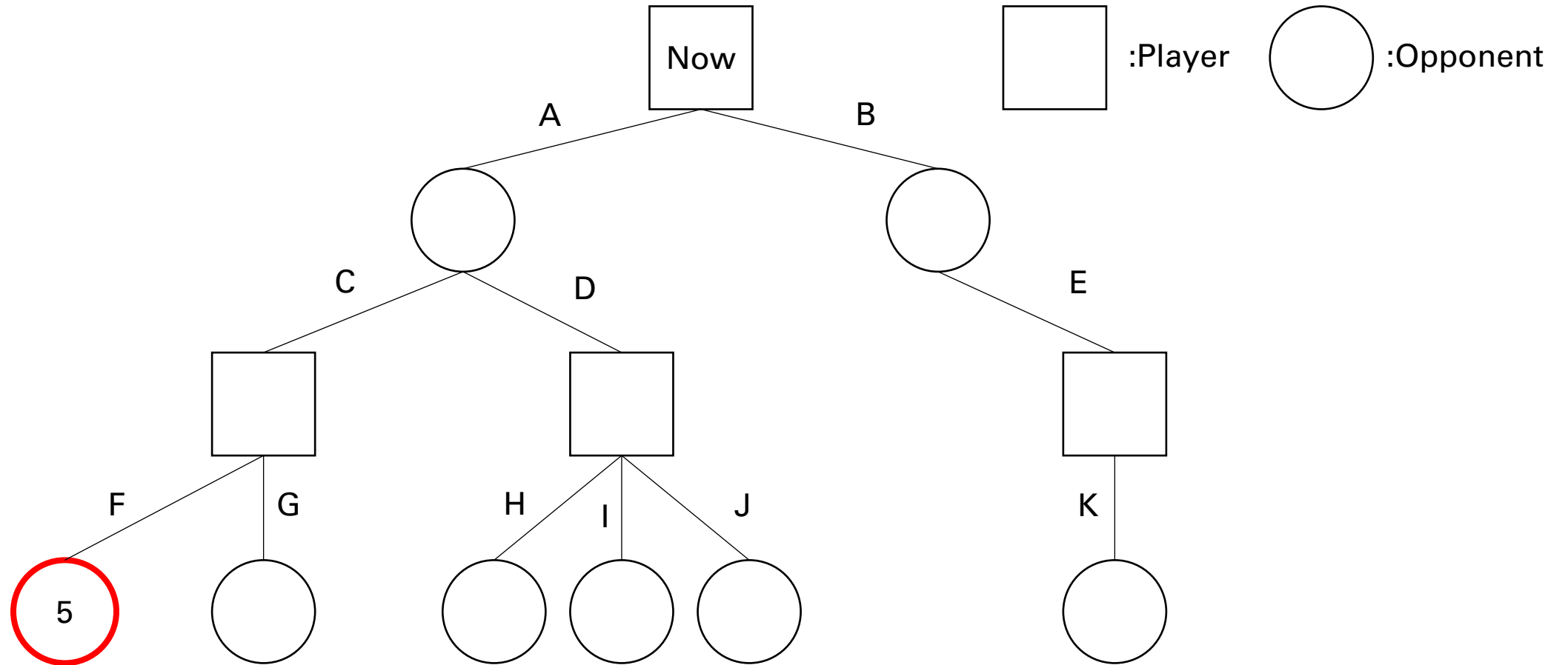
Source:

<https://en.wikipedia.org/wiki/Minimax>

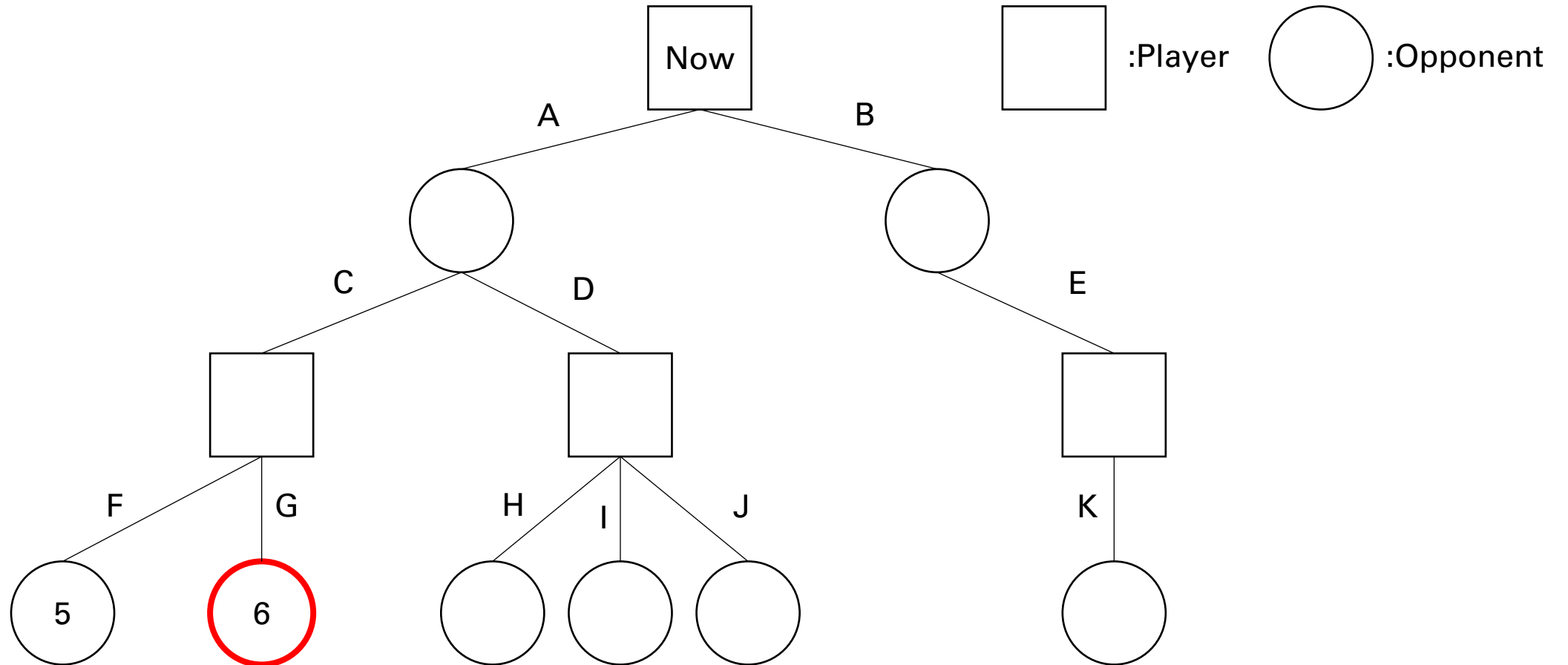
Example



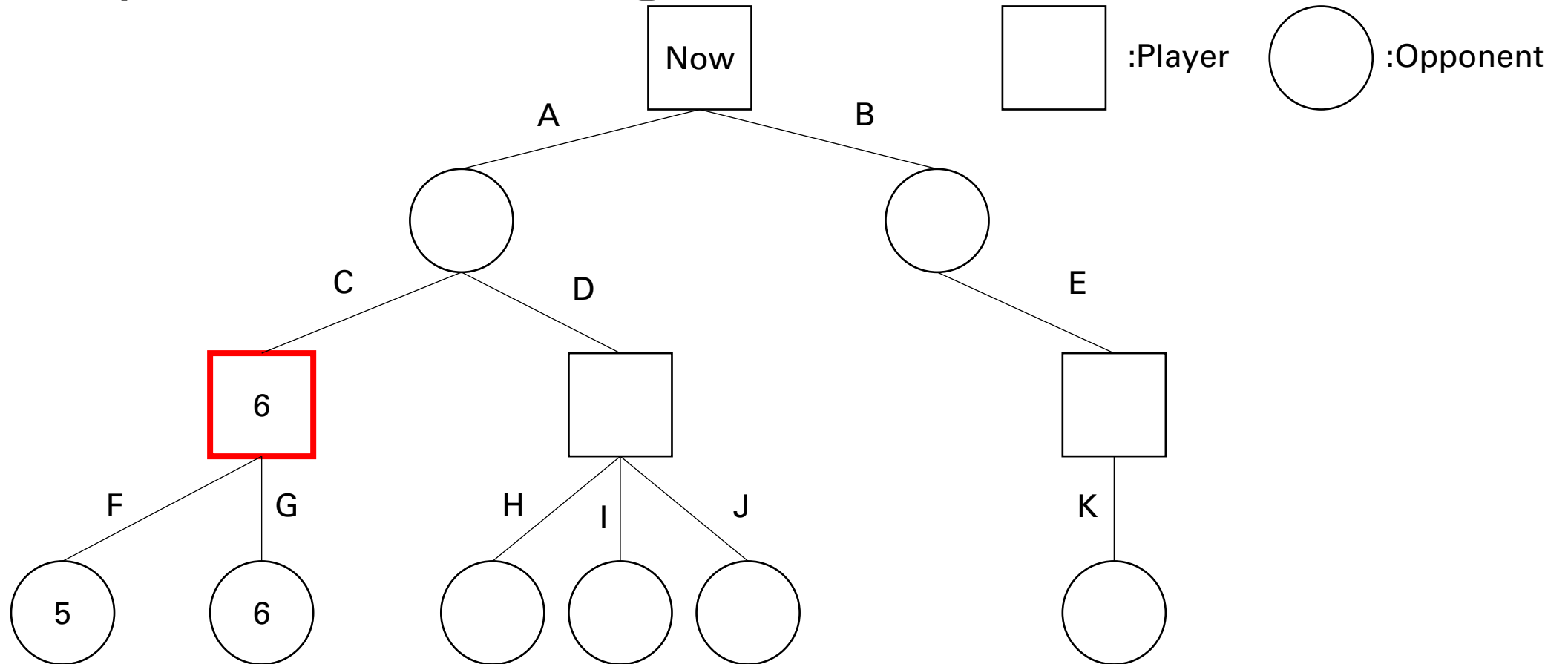
Evaluate score at leaves



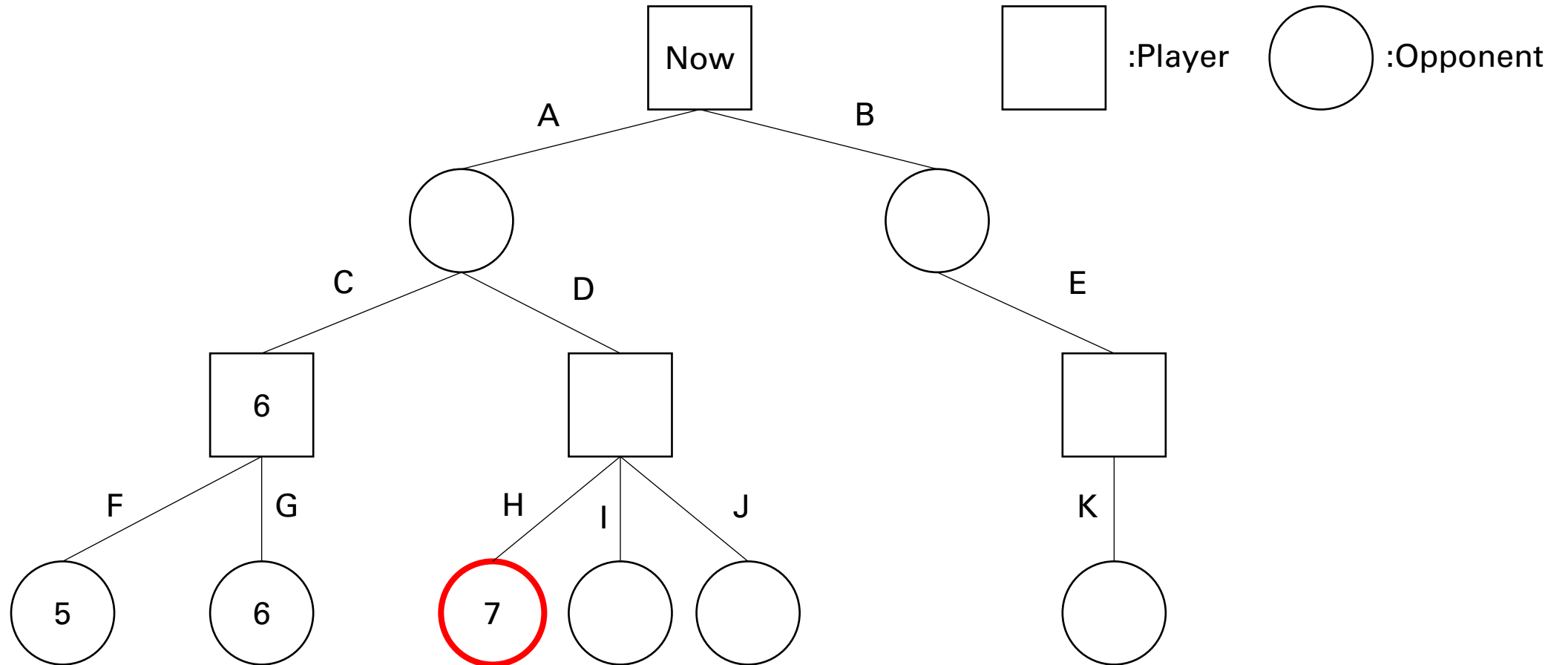
Evaluate score at leaves



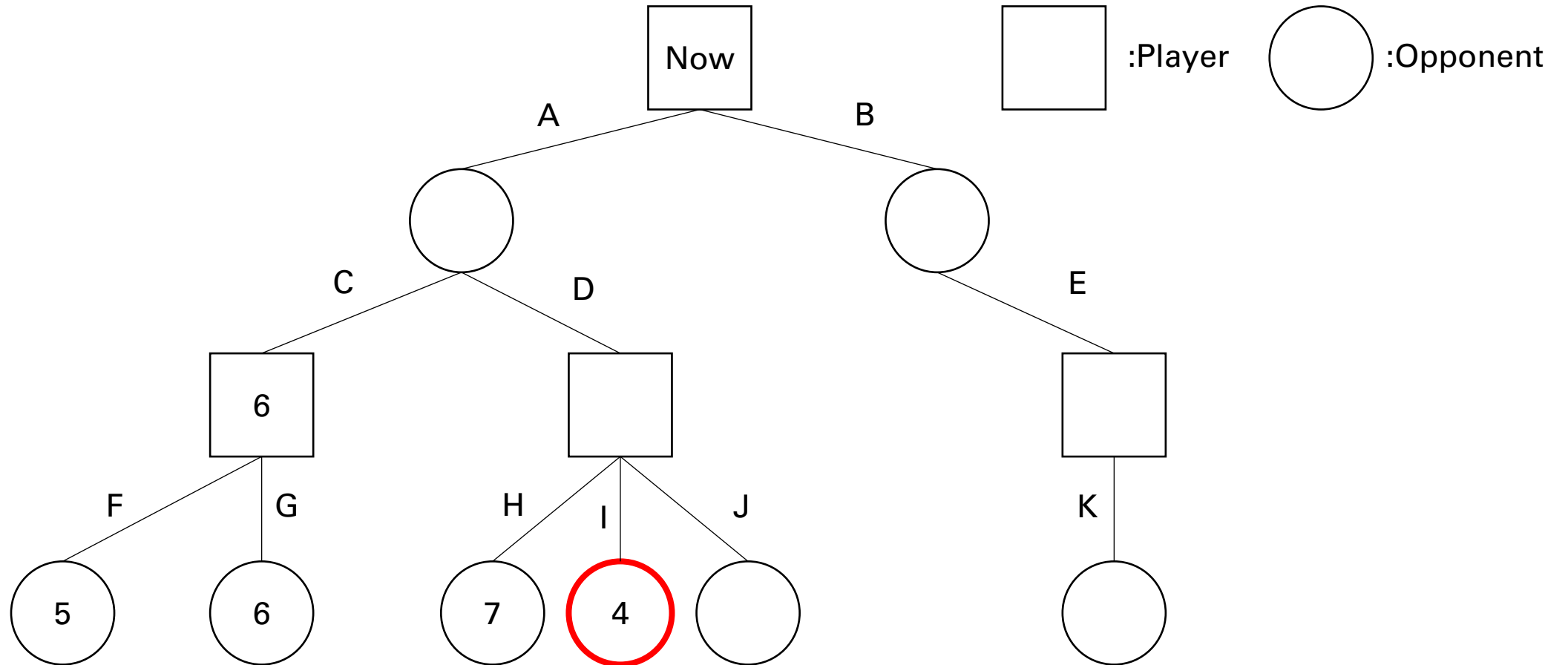
Player picks the largest score



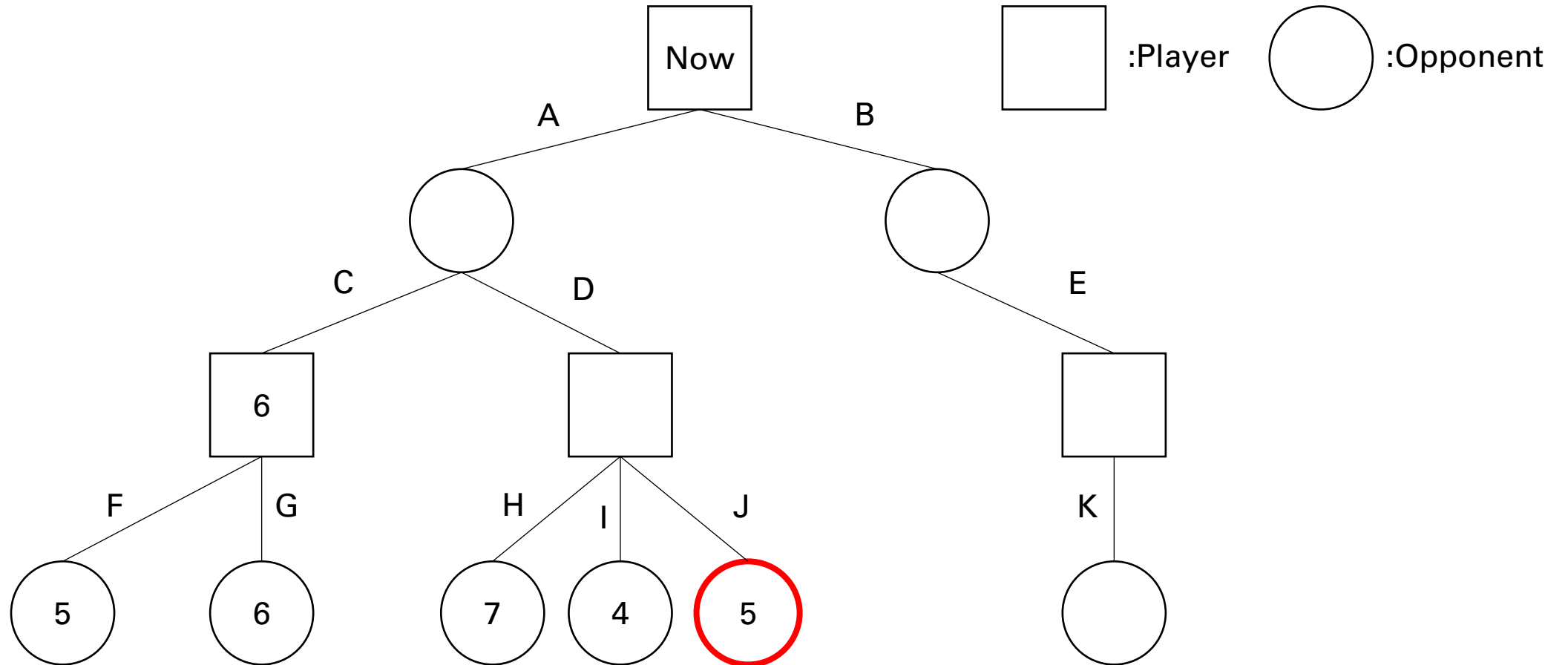
Evaluate score at leaves



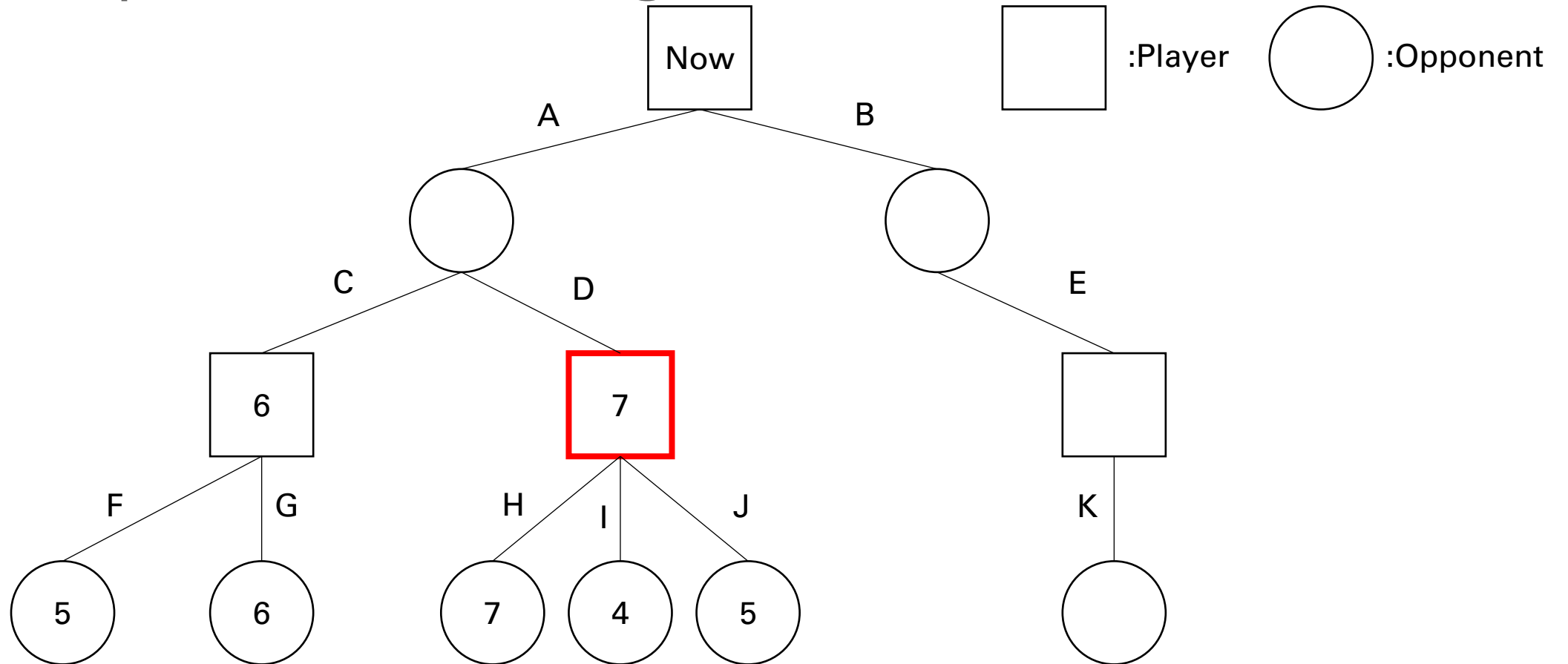
Evaluate score at leaves



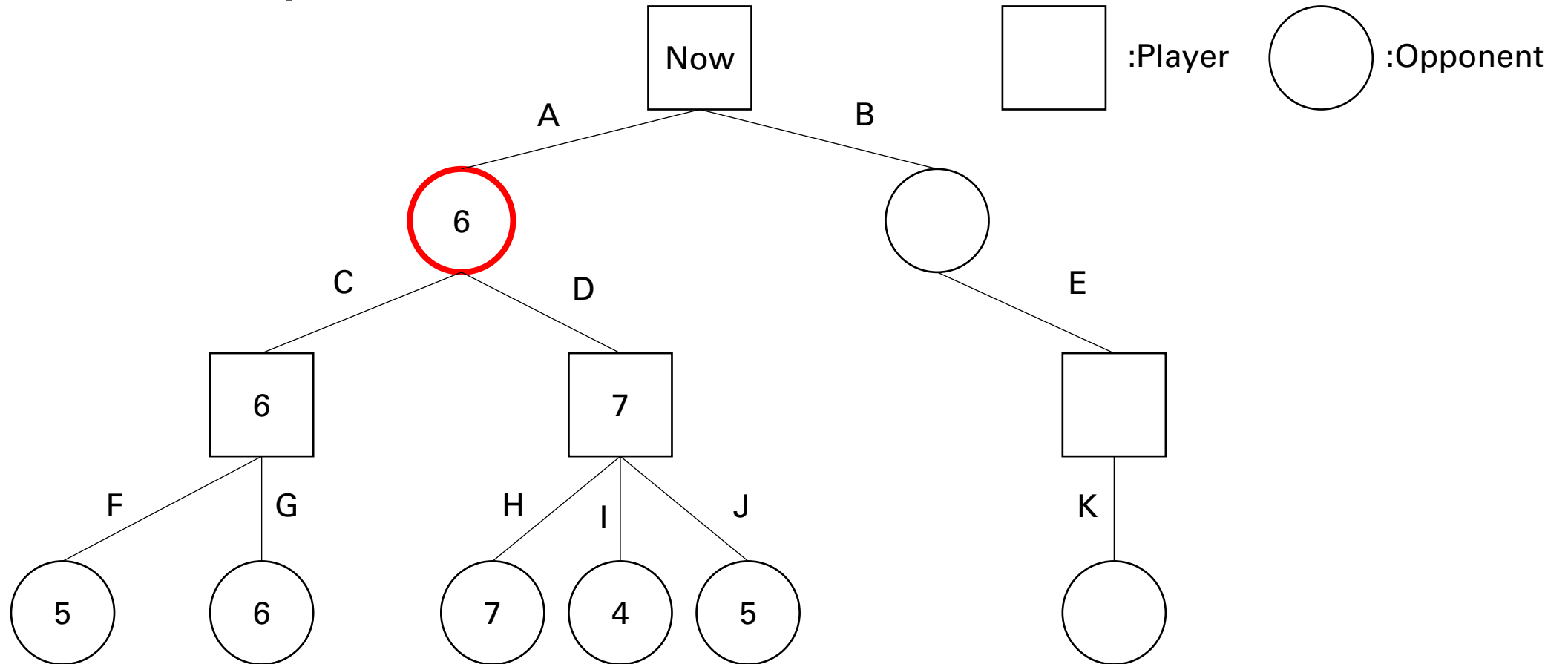
Evaluate score at leaves



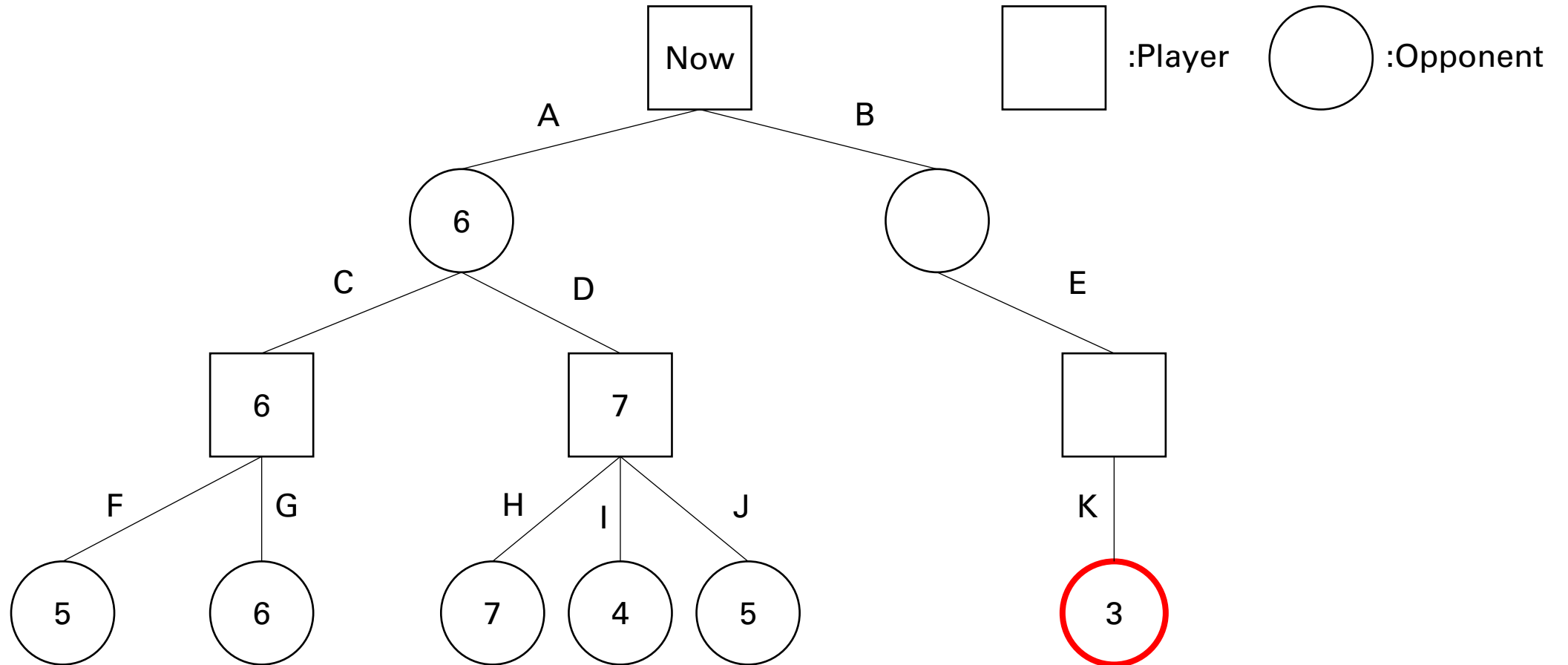
Player picks the largest score



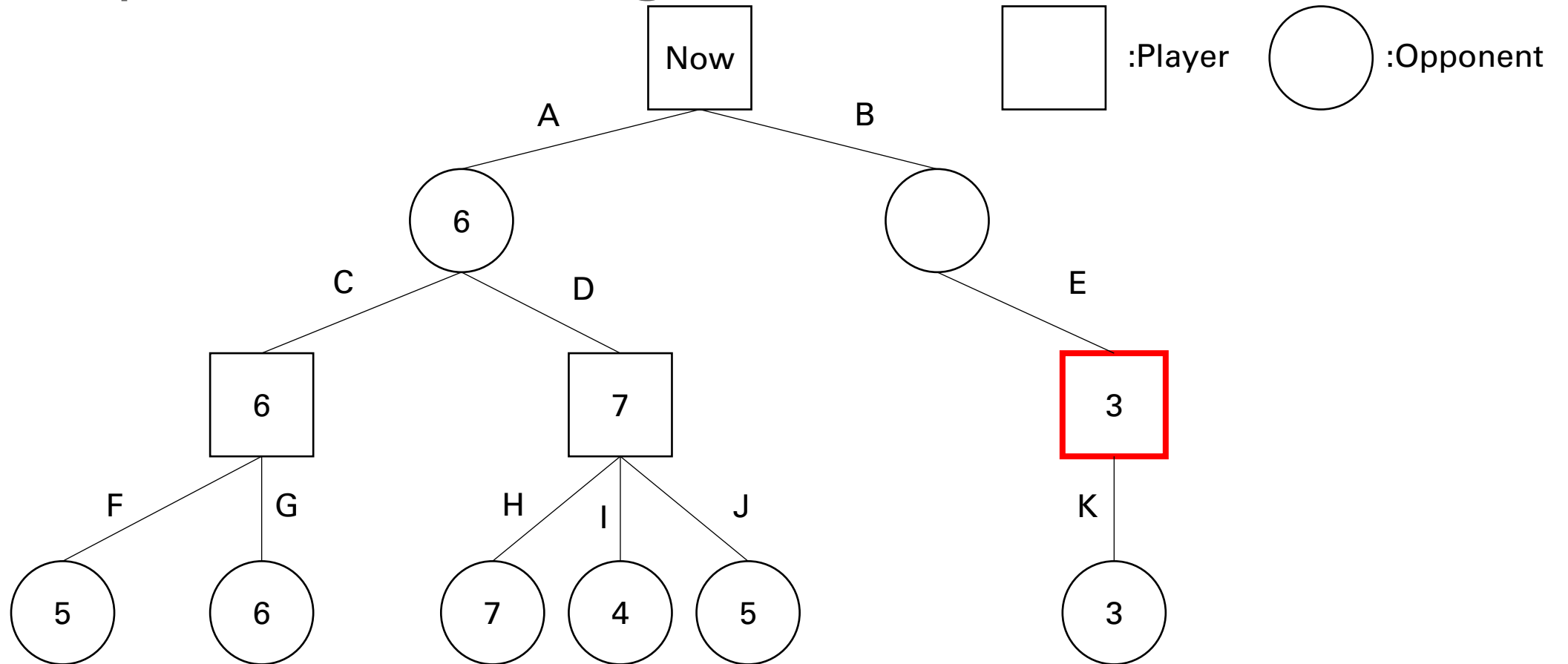
Opponent picks the smallest score



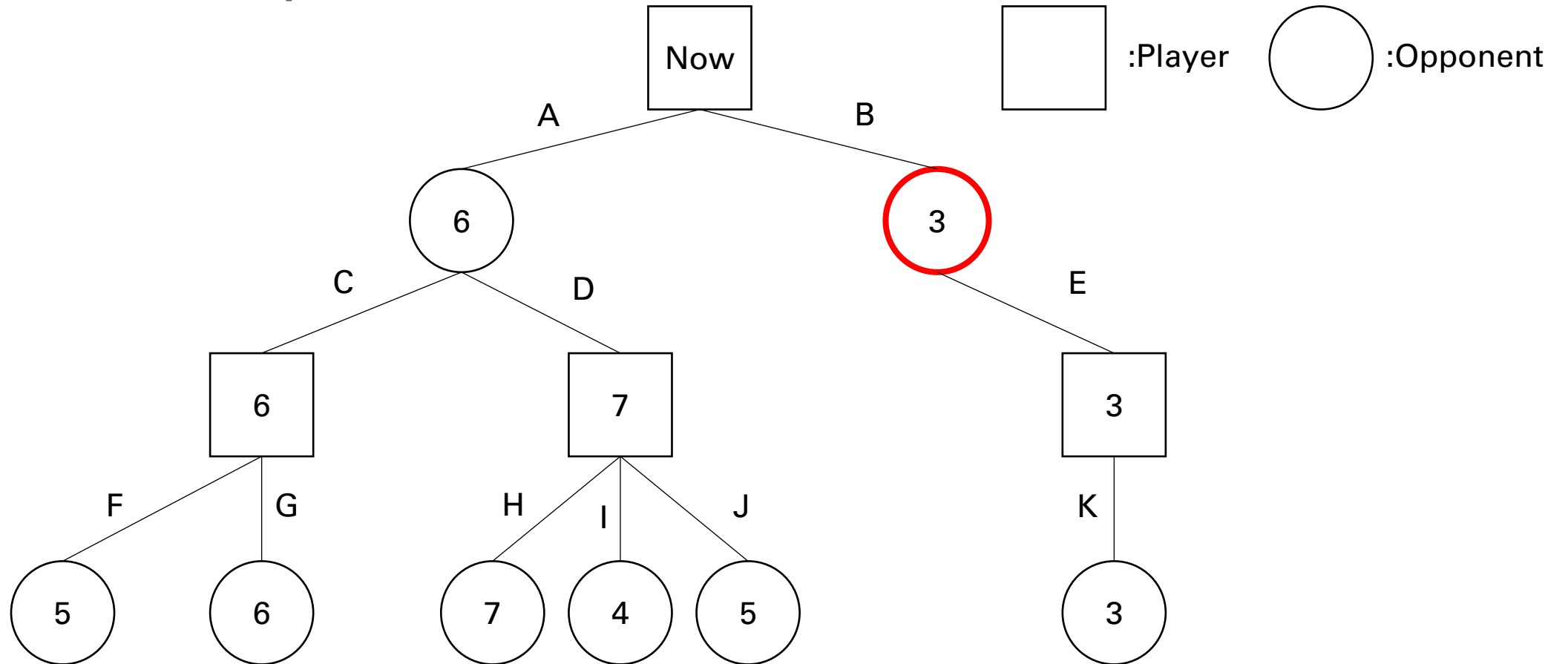
Evaluate score at leaves



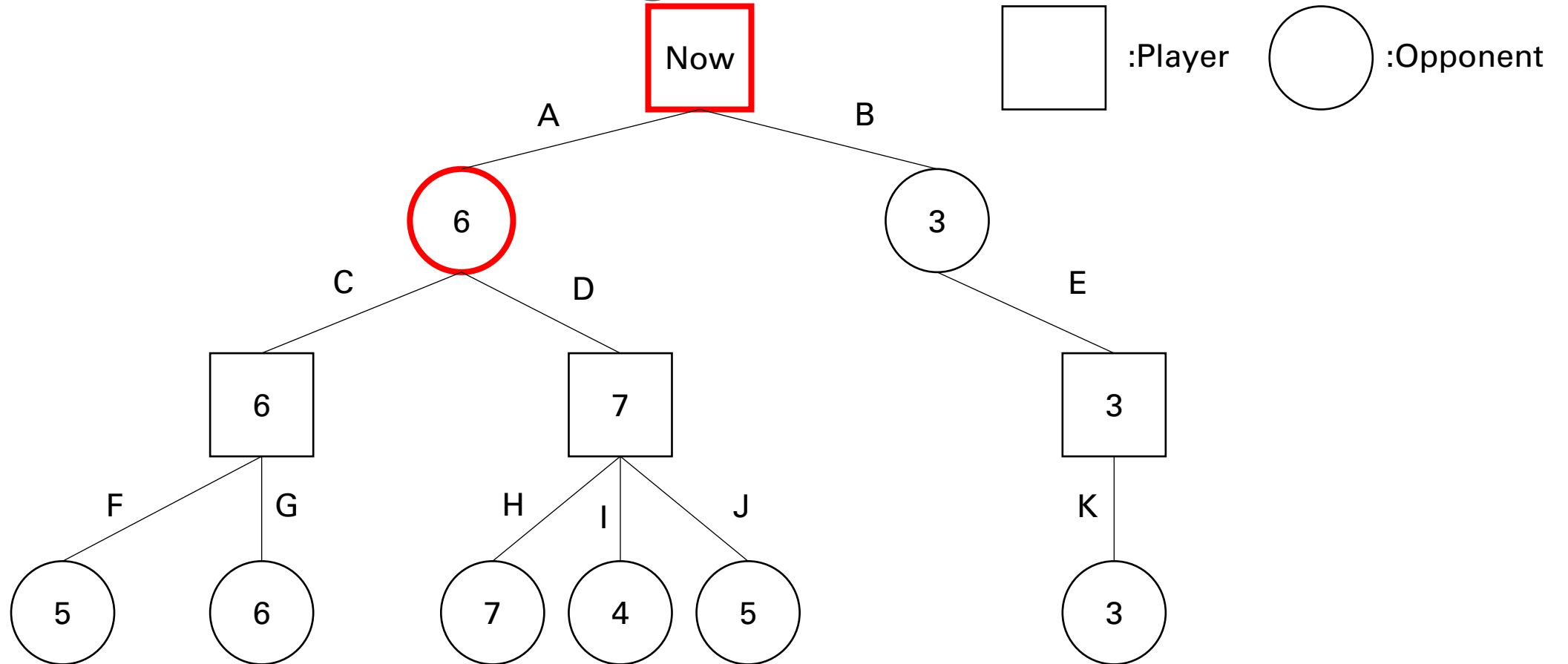
Player picks the largest score



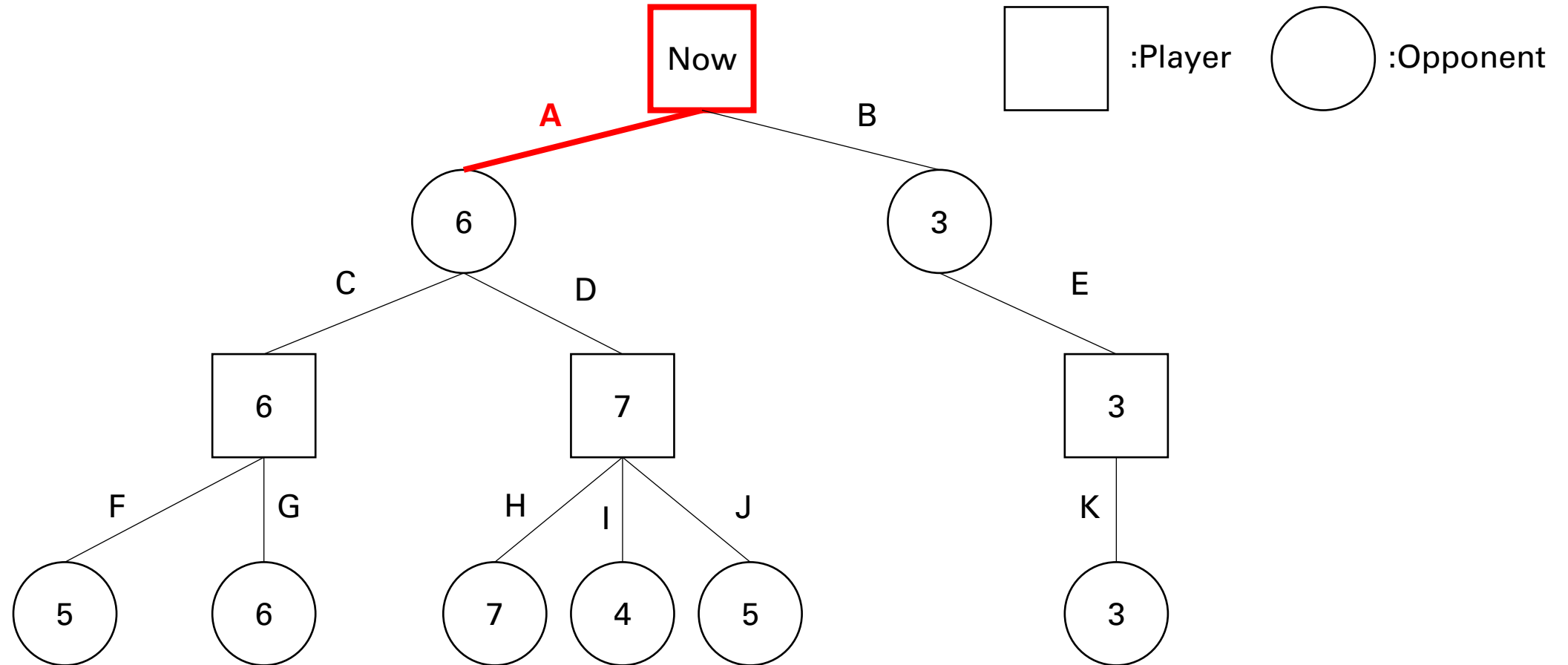
Opponent picks the smallest score



Move A has the largest score



Player picks move A to be the next move



Outline

1. Introduction
2. Chess and Mini Chess
3. State Value Function
4. Minimax
5. **Alpha-Beta Pruning**
6. How To Design Your AI
7. Package
8. Requirements
9. Grading
10. Submission

Alpha-Beta Pruning

- ♦ By Minimax, we can simulate our opponent's moves and pick a move with minimum risk and maximum value
- ♦ Looking forward to more steps that may improve the policy
- ♦ However, the size of the search tree may drastically increase with the increase in 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 are not related to the result
- ♦ We can try to “prune” these branches to improve efficiency
- ♦ The Alpha-Beta Pruning is the improved version of the Minimax method which eliminates some unnecessary branches

Alpha-Beta Pruning Pseudocode

```
function alphabeta(node, depth,  $\alpha$ ,  $\beta$ , maximizingPlayer) is
  if depth = 0 or node is a terminal node then
    return the heuristic value of node
  if maximizingPlayer then
    value :=  $-\infty$ 
    for each child of node do
      value := max(value, alphabeta(child, depth - 1,  $\alpha$ ,  $\beta$ , FALSE))
       $\alpha$  := max( $\alpha$ , value)
      if  $\alpha \geq \beta$  then
        break (*  $\beta$  cutoff *)
    return value
  else
    value :=  $+\infty$ 
    for each child of node do
      value := min(value, alphabeta(child, depth - 1,  $\alpha$ ,  $\beta$ , TRUE))
       $\beta$  := min( $\beta$ , value)
      if  $\beta \leq \alpha$  then
        break (*  $\alpha$  cutoff *)
    return value
```

Source:

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

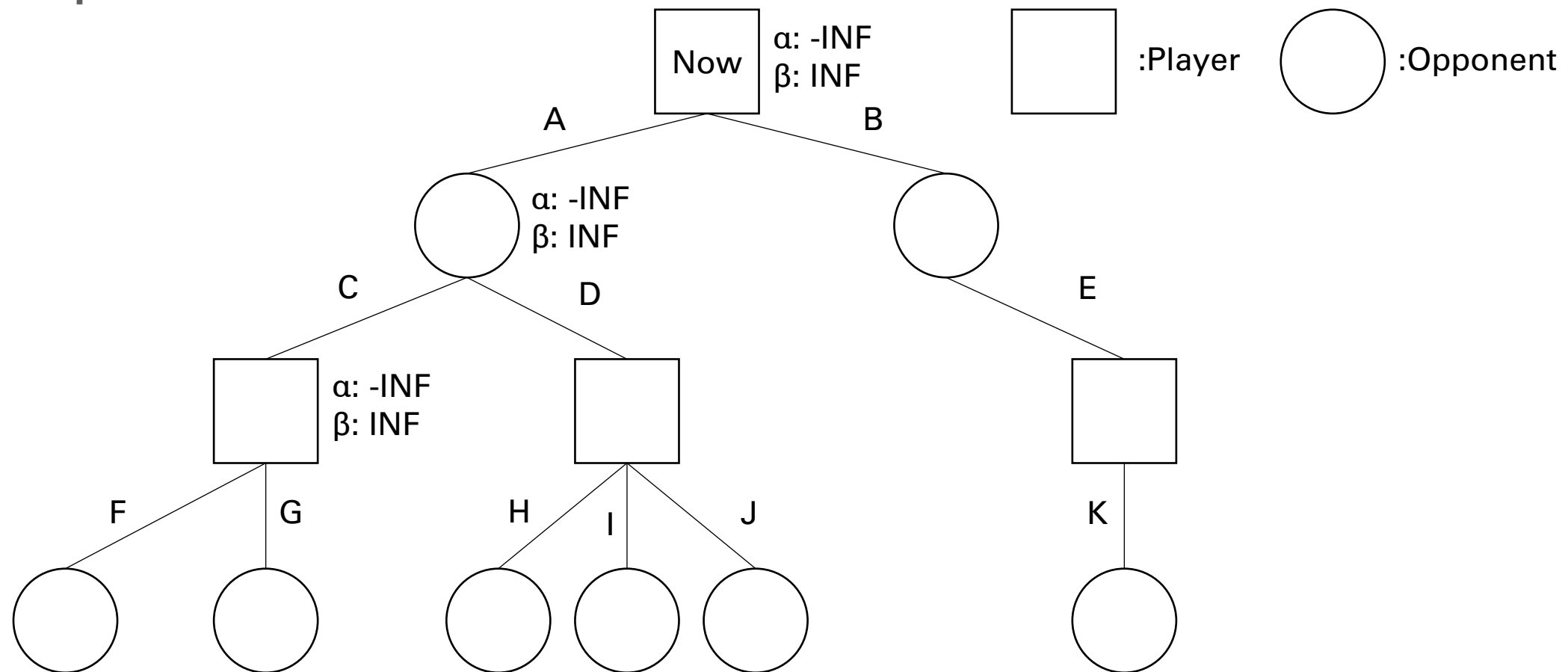
Alpha-Beta Pruning

- ♦ 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

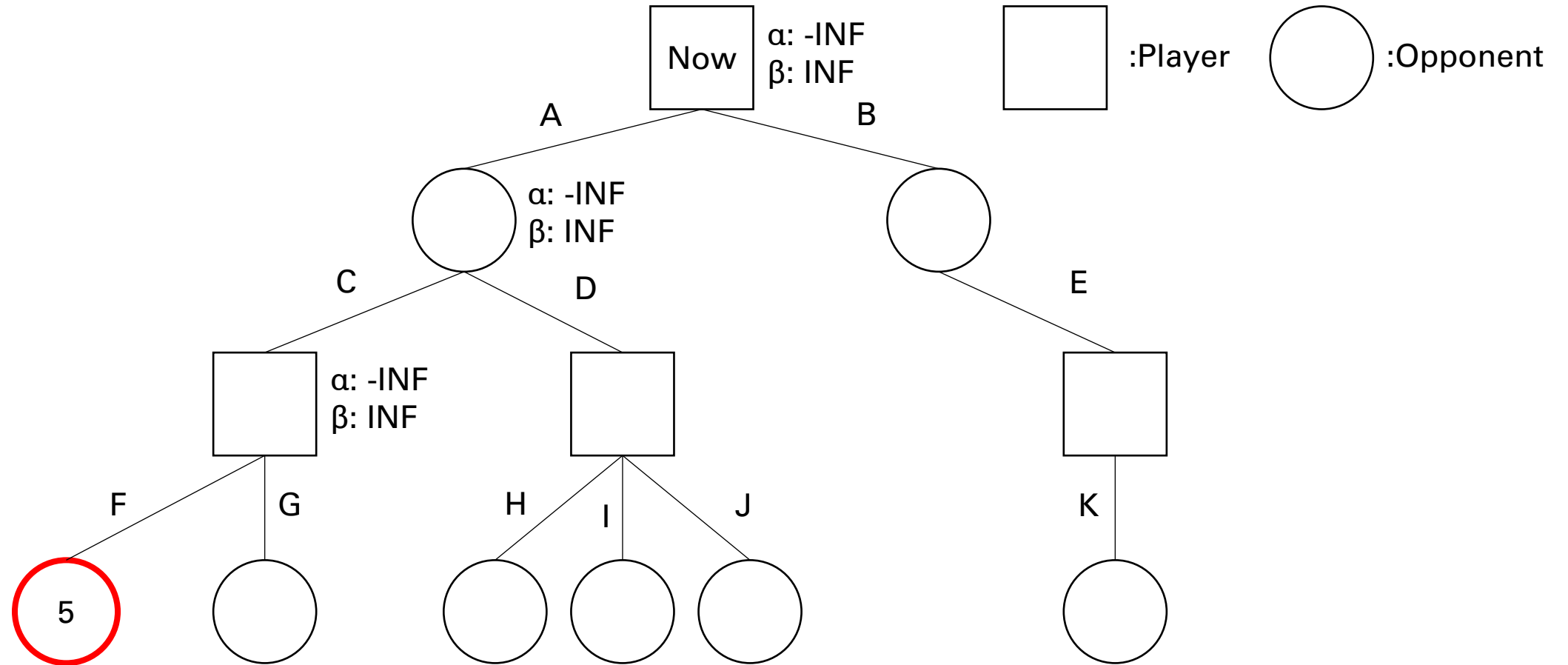
Alpha-Beta Pruning

- ♦ If $\alpha \geq \beta$ on a player node, we can stop to search on this branch
- ♦ In this situation, the player will return a value $\geq \beta$ on this branch
- ♦ However, the opponent already has a better choice (β)
- ♦ 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 $\beta \leq \alpha$ on an opponent node

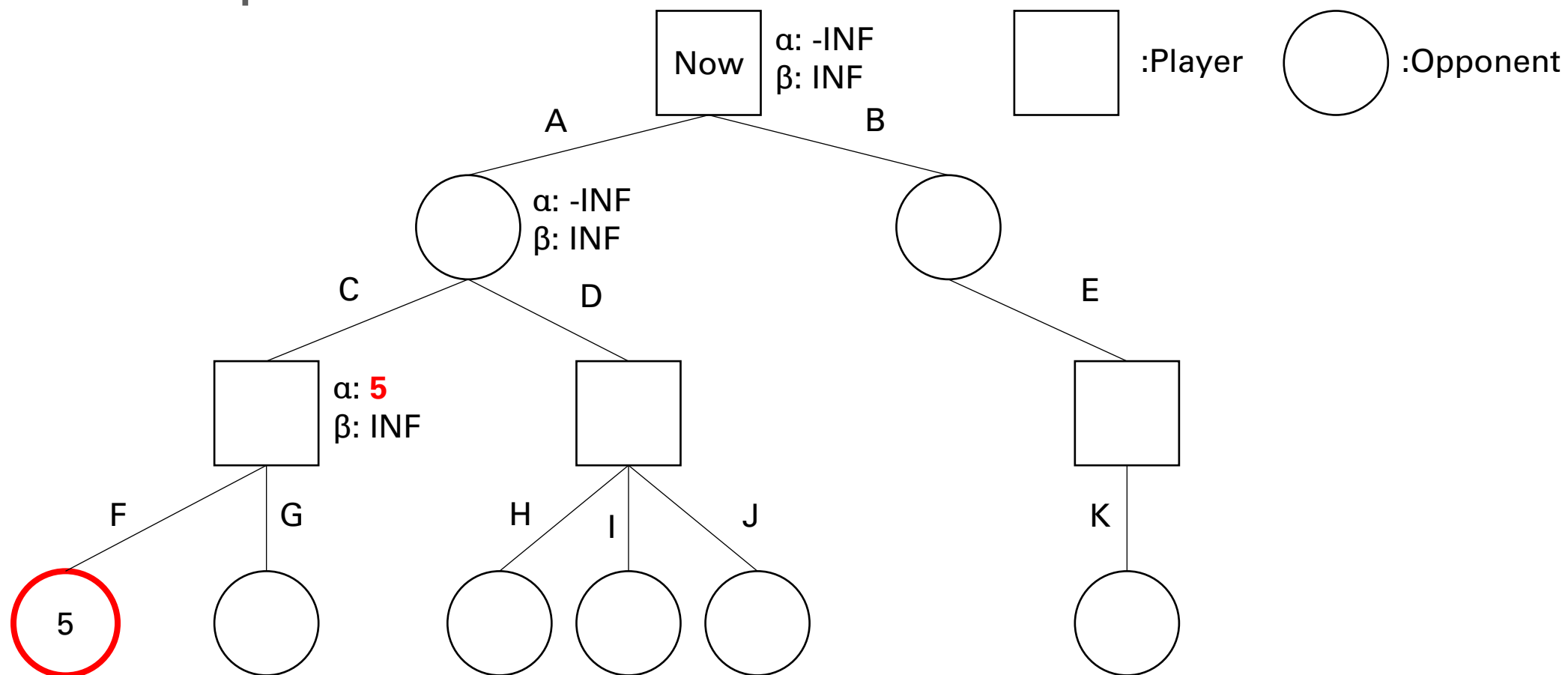
Example



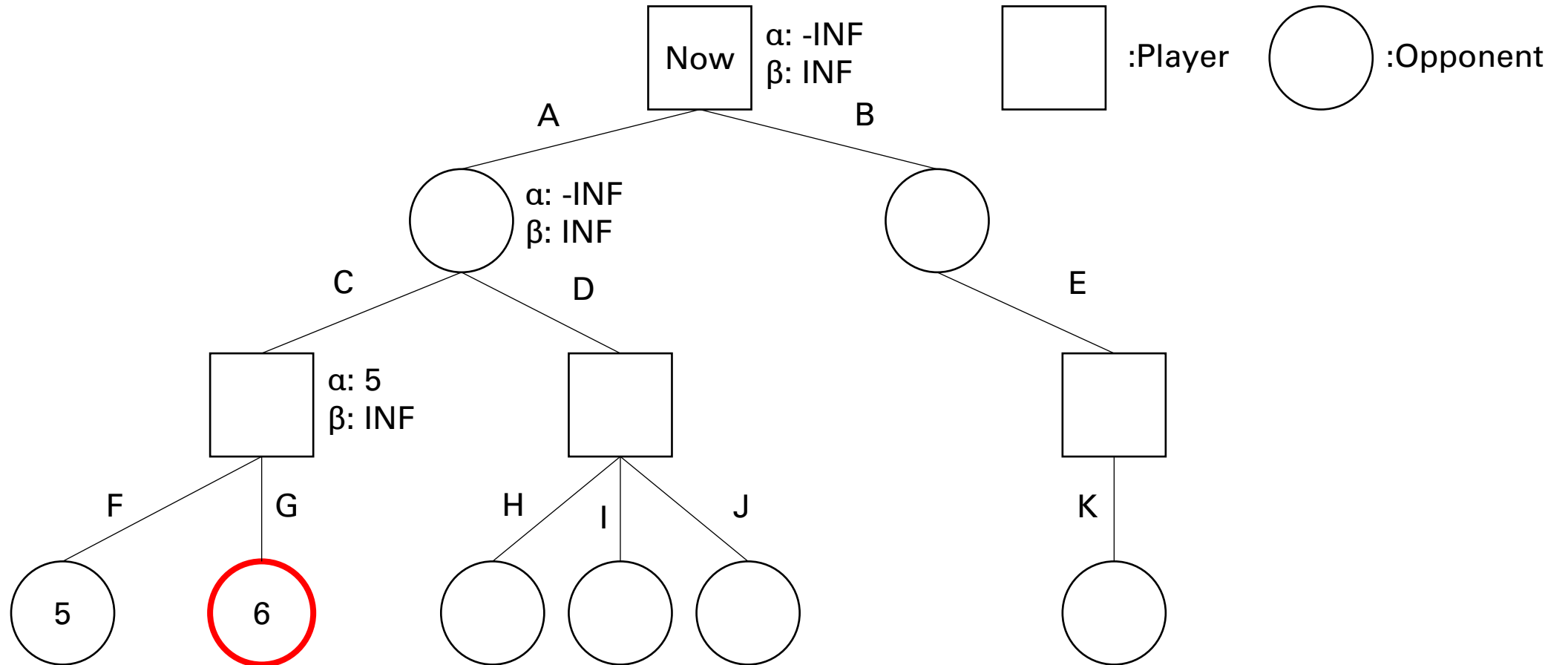
Evaluate score at leaves



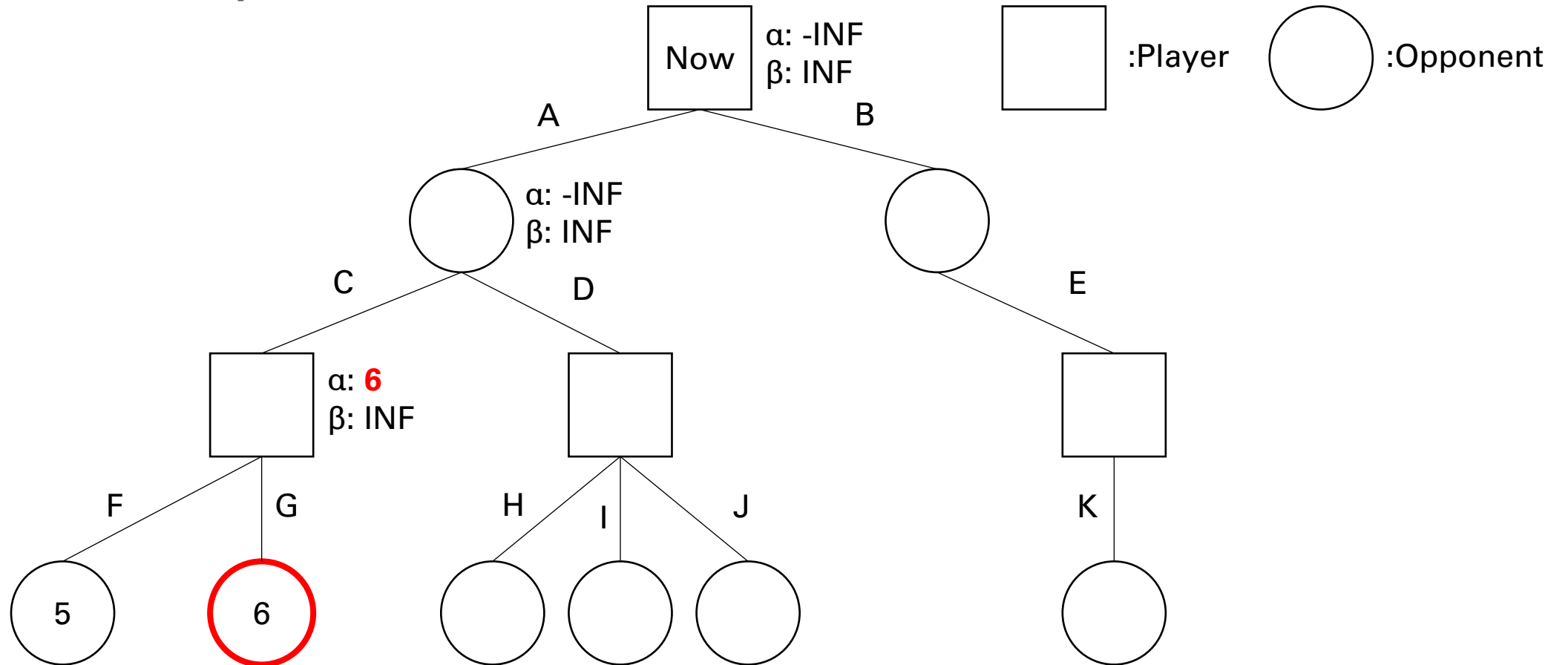
Update alpha



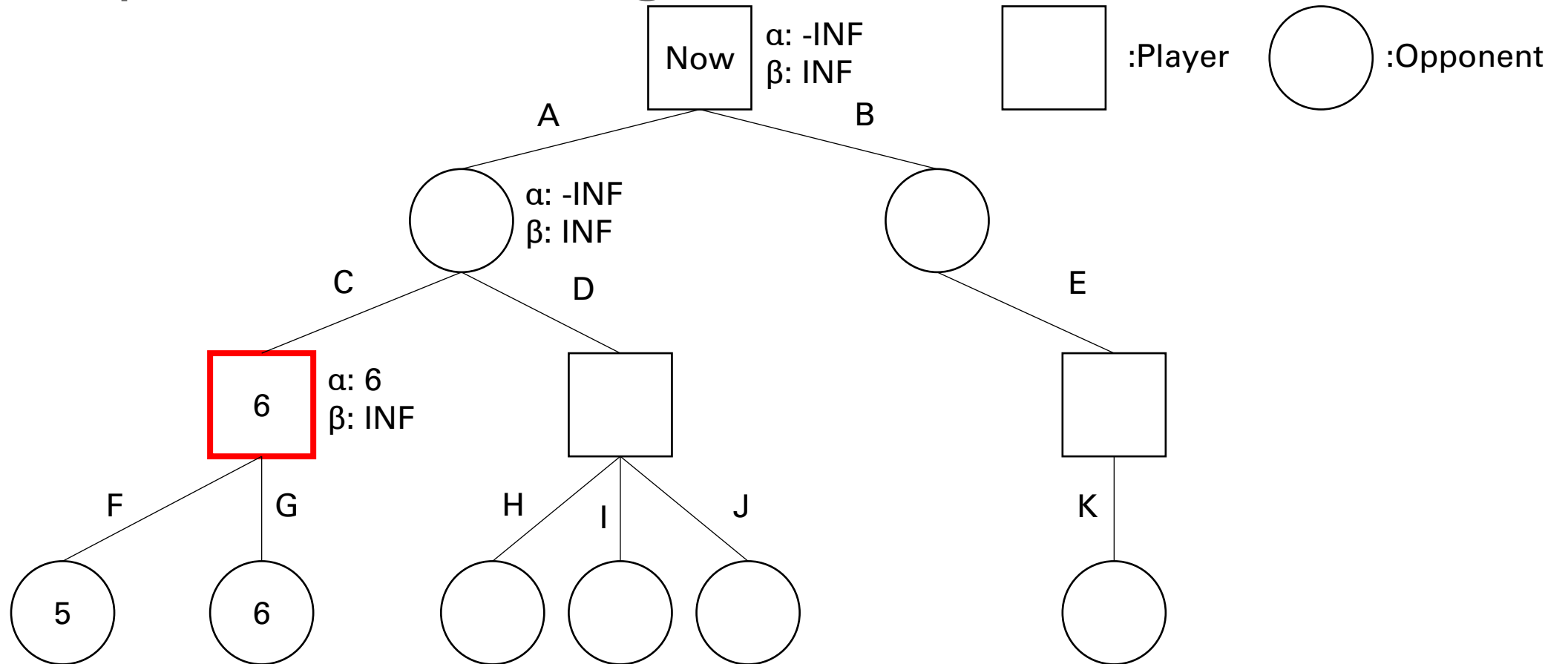
Evaluate score at leaves



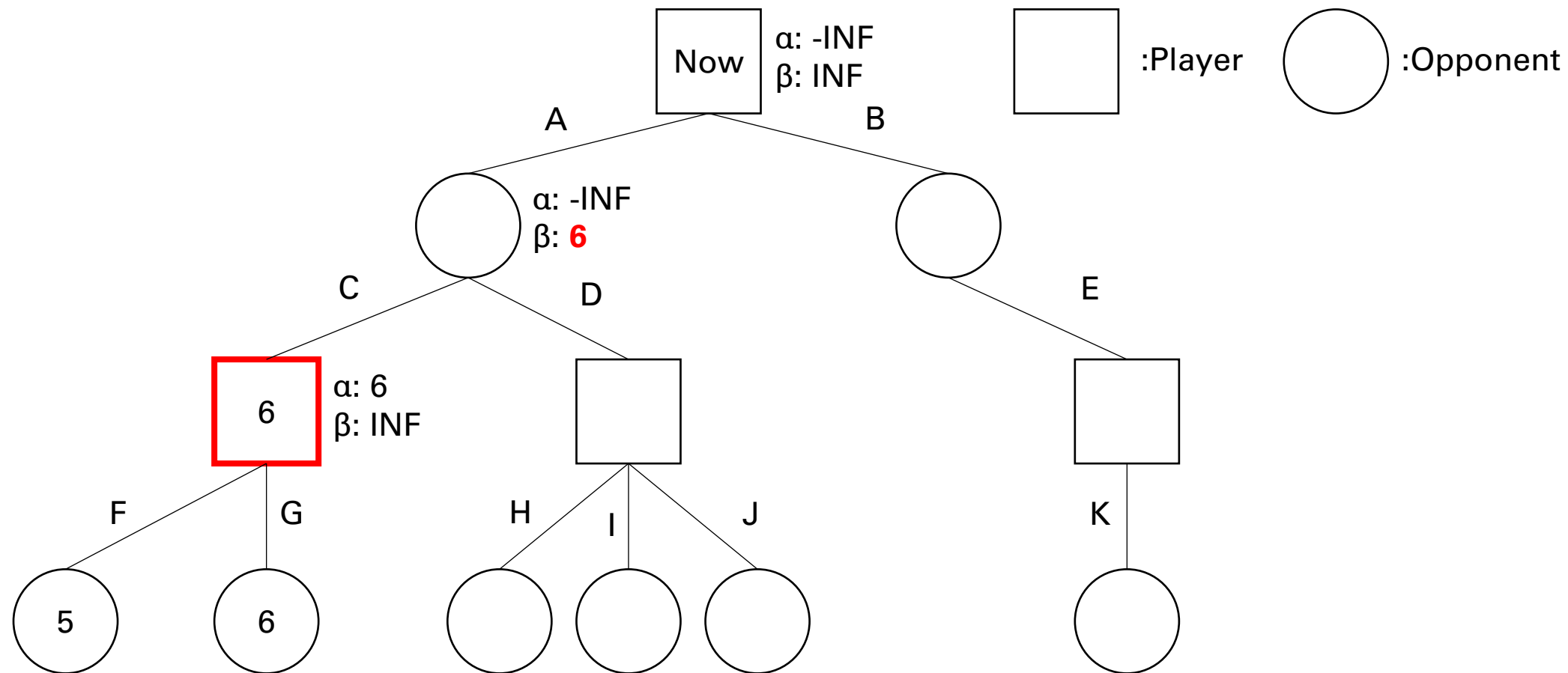
Update alpha



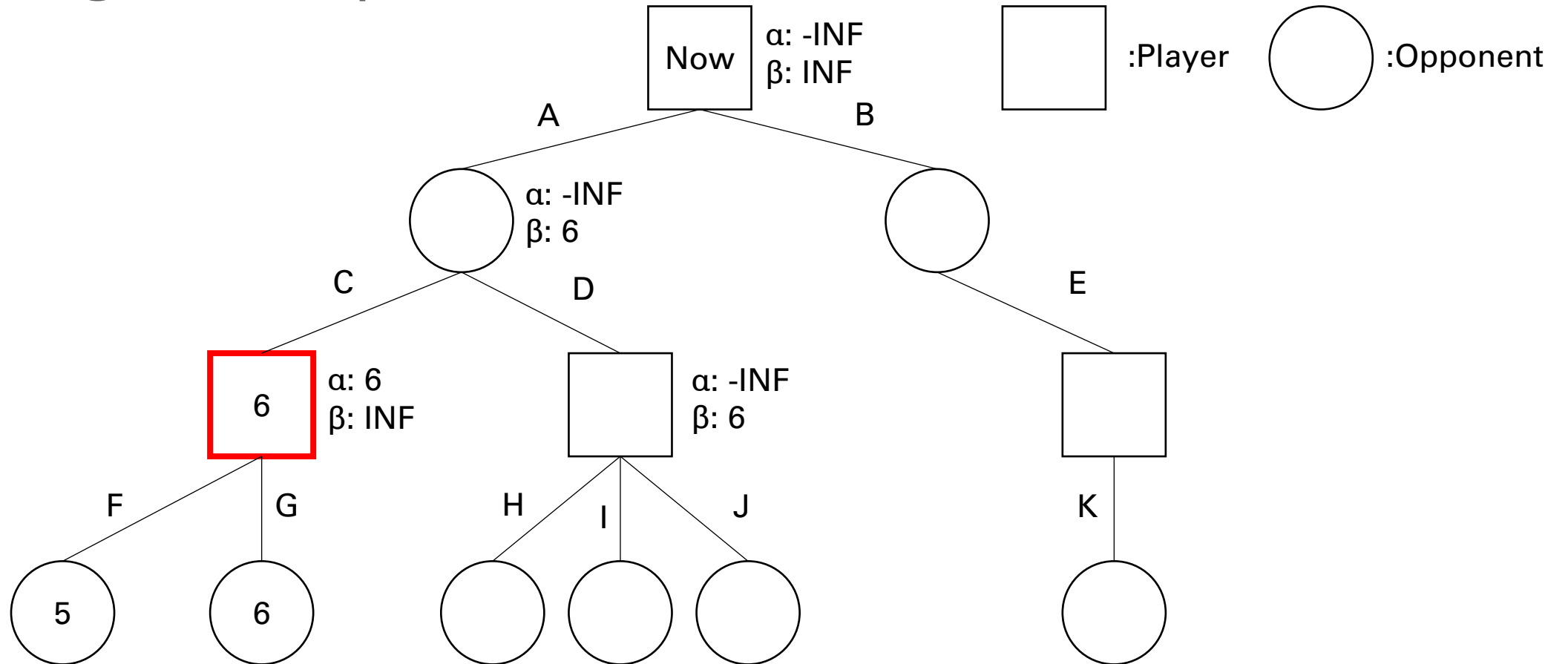
Player picks the largest score



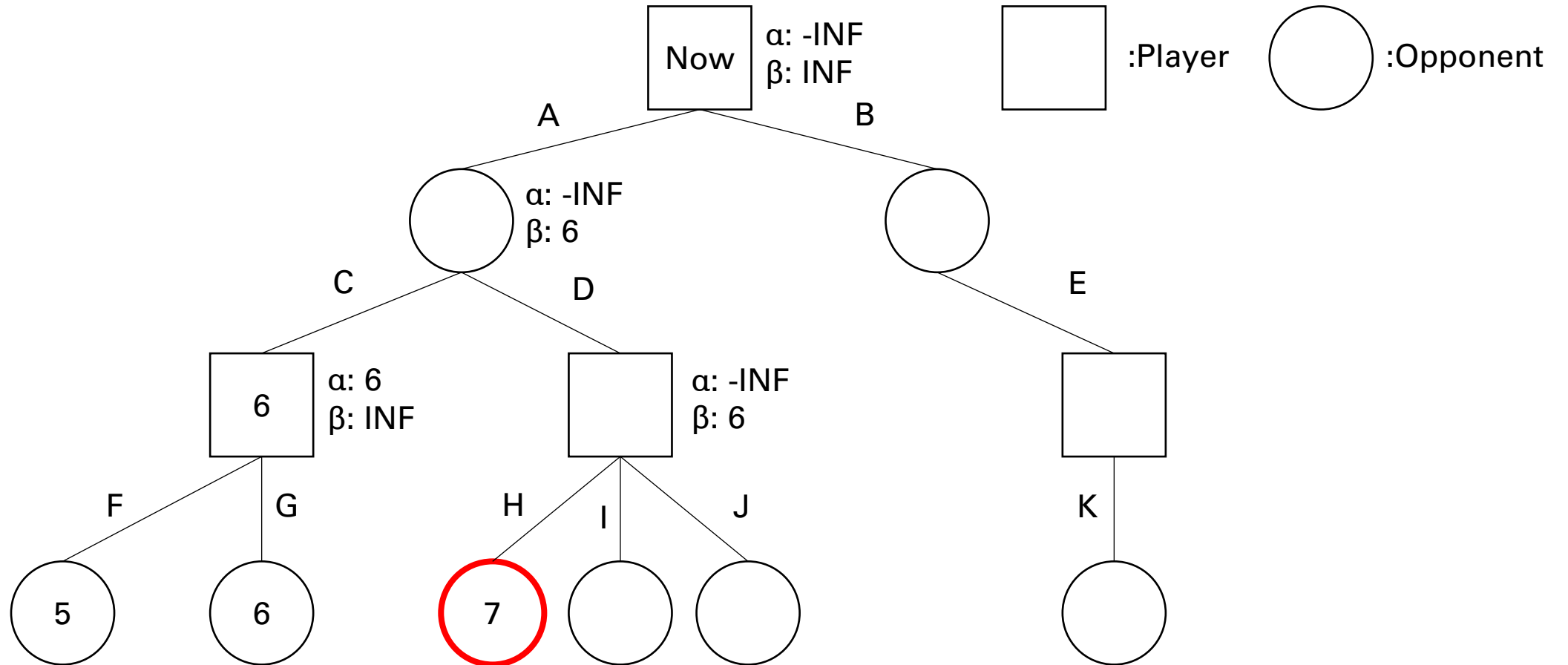
Update beta



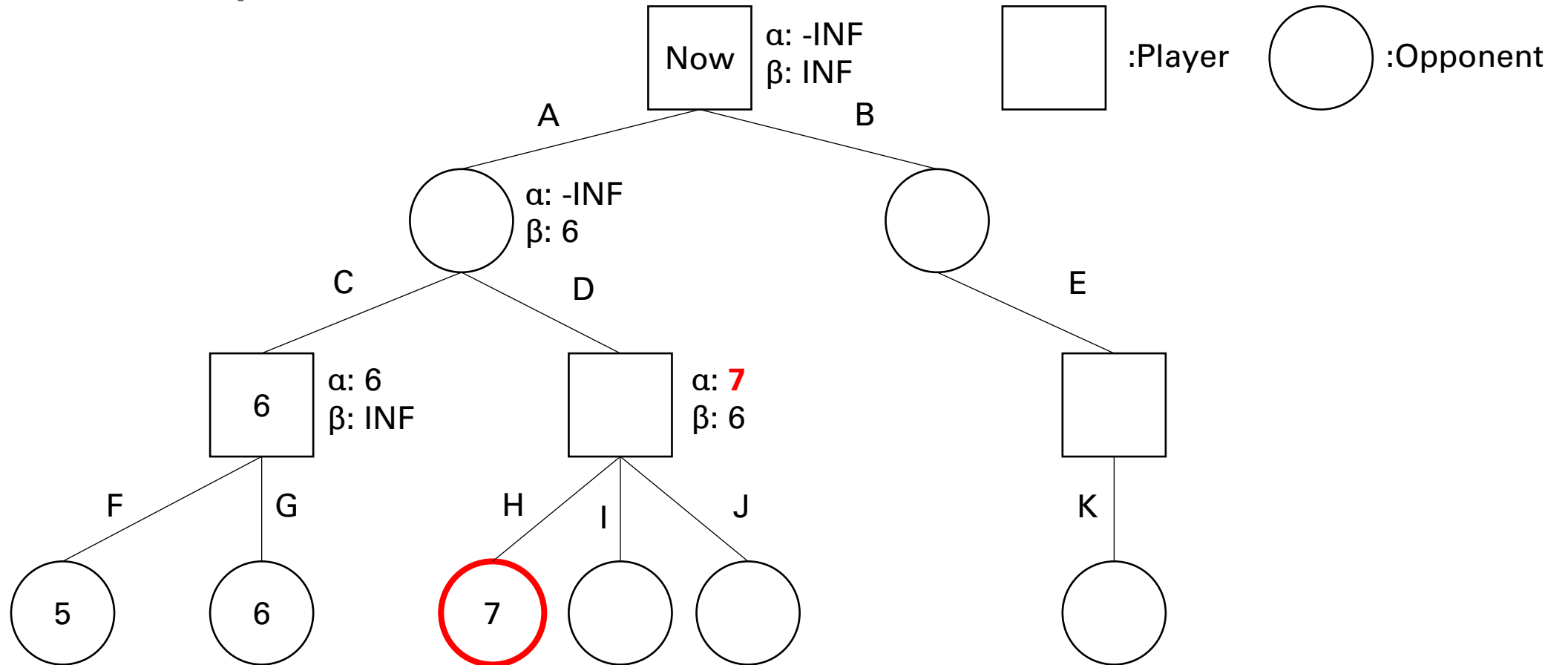
Propagate alpha and beta values



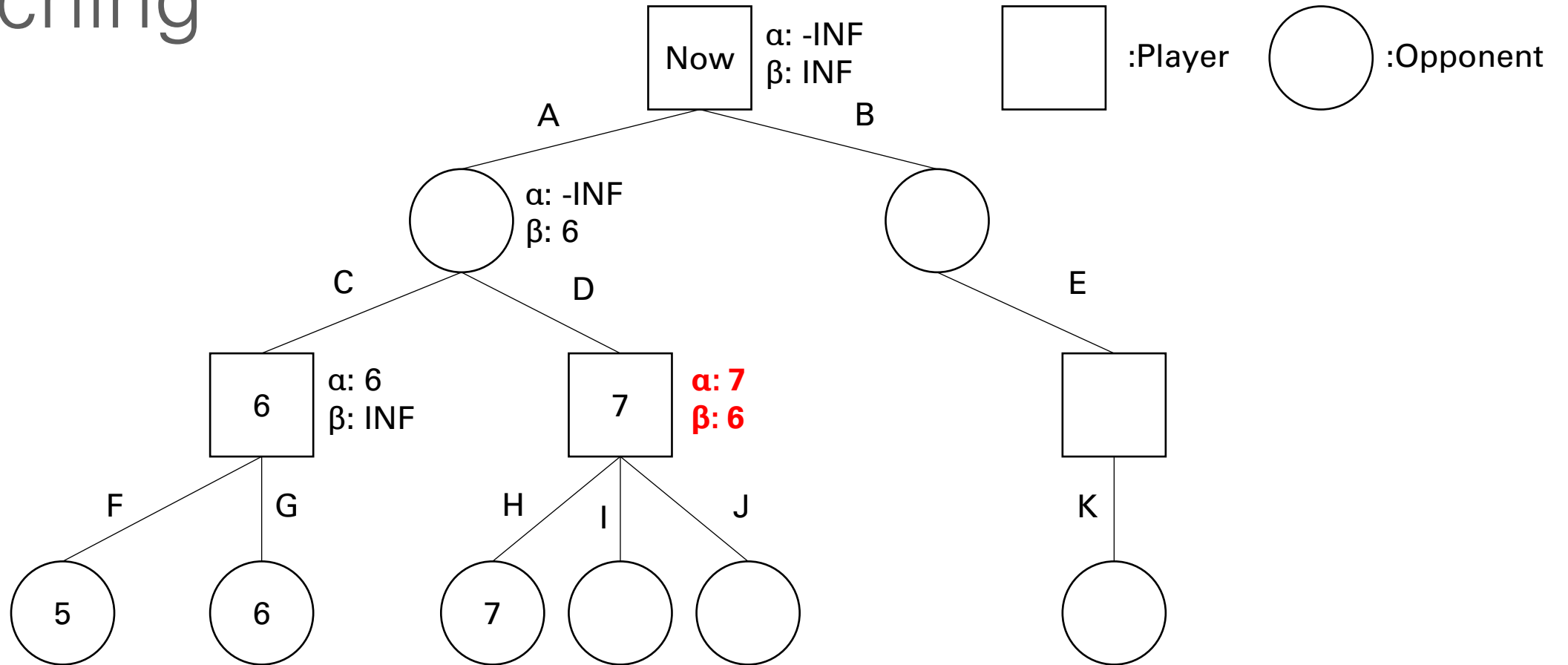
Evaluate score at leaves



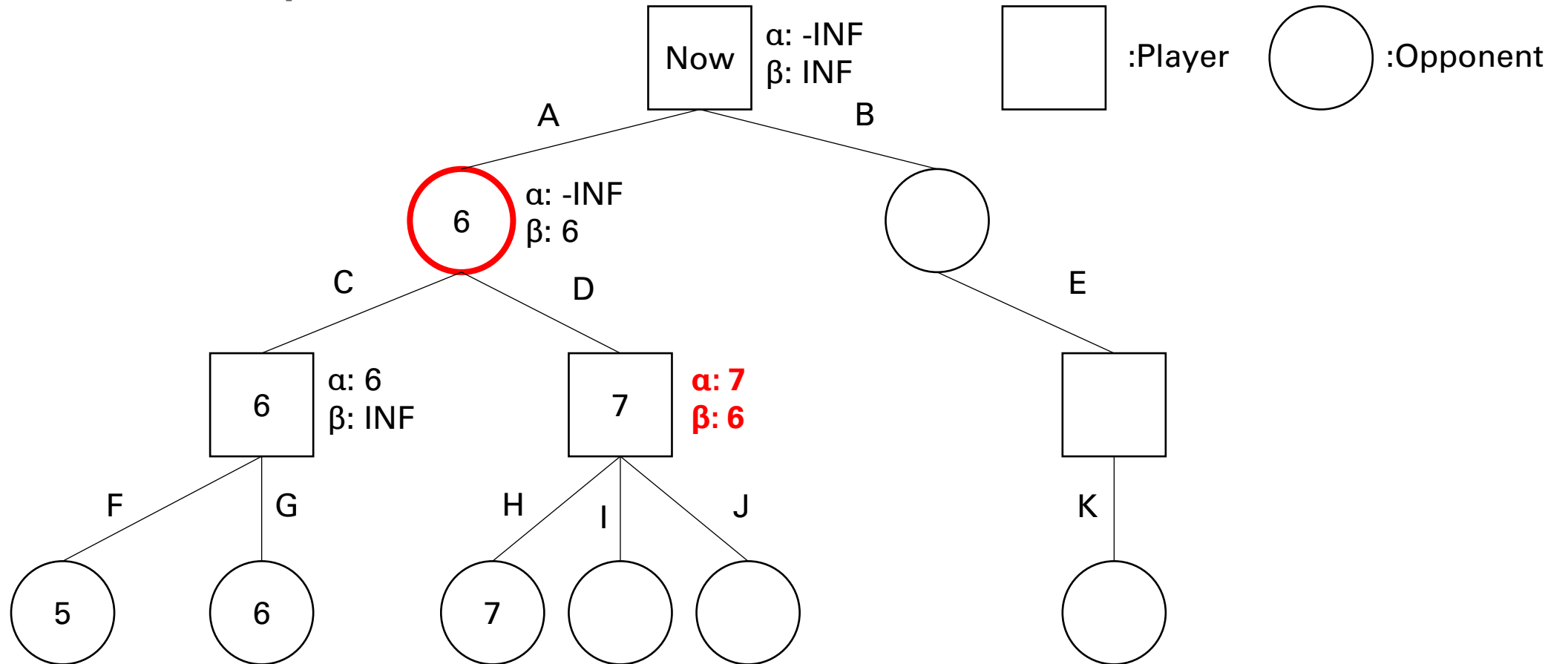
Update alpha



Alpha \geq 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. Chess and Mini Chess
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading
10. Submission

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

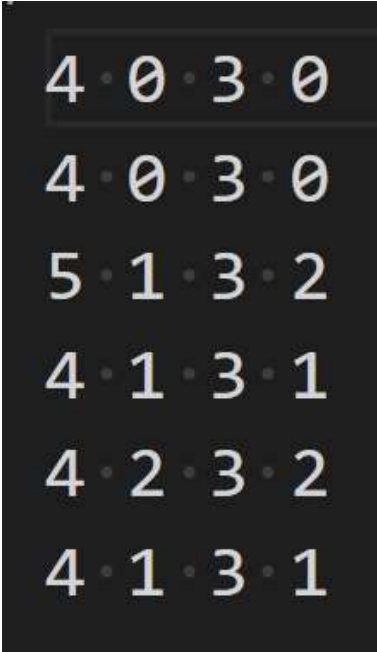
- 3 part
- First is the player (0 for white player, 1 for black player)
- Second part is white player's board (5*6)
- Third part is black player's board (5*6)
- 0=empty, 1=pawn, 2=rook, 3=knight, 4=bishop, 5=queen, 6=king

```
0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
1 1 1 1 1
2 3 4 5 6

6 5 4 3 2
1 1 1 1 1
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 within the time limit
- Only the last complete output will be considered
 - In this case, on the right, 4 1 3 1 will be accepted by the game runner
- You lose if you output an invalid move
- Move format: from.y from.x to.y to.x
 - More details in the package introduction section



4	0	3	0
4	0	3	0
5	1	3	2
4	1	3	1
4	2	3	2
4	1	3	1

How to design your AI

- ♦ You can refer to the random.cpp/hpp in the src/policy and src/player folders
- ♦ Design your state value function in state.cpp 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. Chess and Mini Chess
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. **Package**
8. Requirements
9. Grading
10. Submission

Package

- ♦ You don't need to implement all the things by yourself, we will provide some useful utilities, and then you can focus on the state value function and the tree search algorithm.
- ♦ You will get:
 - ♦ Game runner
 - ♦ State class
 - ♦ a native method to get all legal actions
 - ♦ Generate a new state based on action and state
 - ♦ You need to simply state value function by yourself
 - ♦ A example player (with random choose policy)

Package – How to run it

- ♦ Despite the source files, you will also get some additional files:
 - ♦ Makefile – help you to compile the project
 - ♦ .gitignore – since you need to push your code to github, this can help you ignore some files when you push it
- ♦ You can modify your Makefile, but you should make sure it can compile on TAs environment.
- ♦ With Makefile and make utils (more details in environment setup document), you can compile your code with: **make all**

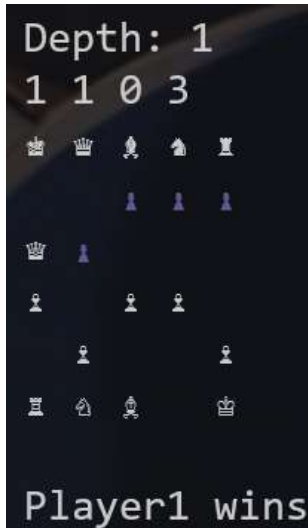
```
>>> make all
g++ --std=c++20 -Wall -Wextra -O3 -o build/main.exe src/state/state.cpp src/main.cpp
g++ --std=c++20 -Wall -Wextra -O3 -o build/player_alphabeta.exe src/state/state.cpp src/policy/alphabeta.cpp src/player/alphabeta.cpp
g++ --std=c++20 -Wall -Wextra -O3 -o build/player_minimax.exe src/state/state.cpp src/policy/minimax.cpp src/player/minimax.cpp
g++ --std=c++20 -Wall -Wextra -O3 -o build/player_random.exe src/state/state.cpp src/policy/random.cpp src/player/random.cpp
```

Package – How to run it

- After running the make all command and compile your program successfully, you can use this command to run the game:

```
./build/main ./build/player_random ./build/player_random|
```

- And it will start running!



Depth: 1
1 1 0 3
♔ ♔ ♚ ♚ ♚
♚ ♚ ♚
♔ ♚
♚ ♚ ♚
♚ ♚ ♚
♚ ♚ ♚
♚ ♚ ♚
♚ ♚ ♚
Player1 wins

Package – State

- And now, you should start your project.
- I recommend you to check state class at first
- next_state can generate new state based on a move
- get_legal_actions will generate all legal actions of this state and store them in legal_actions.
- evaluate is the state value function you need to implement

```
class State{
public:
    GameState game_state = UNKNOWN;
    Board board;
    int player = 0;
    int score = -10000000;
    std::vector<Move> legal_actions;

    State();
    State(int player): player(player){};
    State(Board board): board(board){};
    State(Board board, int player): board(board),
    {
        State* next_state(Move move);
        void get_legal_actions();
        int evaluate();
        void print();
        std::string encode_output();
        std::string encode_state();
    };
};
```

Package - Policy

- If this project, you should implement your own MiniMax and AlphaBeta-pruning policy. And you will get a random policy player for example
- Random policy: State in, random legal action out:

```
/**
 * @brief Randomly get a legal action
 *
 * @param state Now state
 * @param depth You may need this for other policy
 * @return Move
 */
Move Random::get_move(State *state, int depth){
    if(!state->legal_actions.size())
        state->get_legal_actions();

    auto actions = state->legal_actions;
    return actions[(rand()+depth)%actions.size()];
}
```

Package - player

- ♦ There are some useful functions in example random player file:
 - ♦ read_board: read the board from state file
 - ♦ write_valid_sopt: in random player, it will output random point to action file, you can just modify this function to output the point you want.

Outline

1. Introduction
2. Chess and Mini Chess
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. **Requirements**
9. Grading
10. Submission

Requirements - code

- ♦ In this project you should do these things:
- ♦ 1, design your own state value function
- ♦ 2, implement MiniMax and AlphaBeta pruning algorithm
- ♦ 3, utilize your algorithm and state value function to make a strong AI

Requirements - code

- ♦ If you are not satisfied with the Alpha-Beta Pruning algorithm, you can try some more advanced methods (MCTS, NNUE). However, make sure you can explain how Minimax and Alpha-Beta Pruning works during the demo.
- ♦ If you cannot complete the Alpha-Beta Pruning algorithm, implementing the basic Minimax algorithm also gives you some scores.

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 for a limited time. Only the last move is used by the game runner.

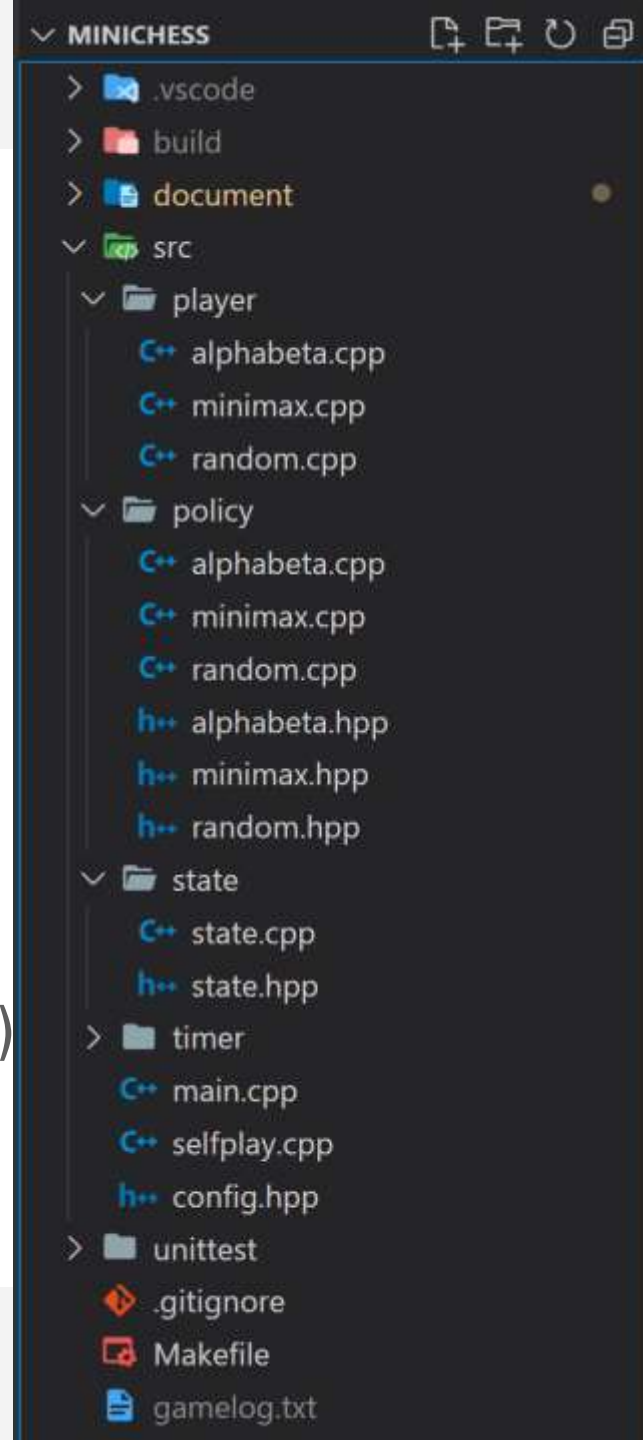
Requirements - code

- ♦ You cannot use these things in your code:
 - ♦ Third party library (only standard library are acceptable)
 - ♦ Inline ASM
 - ♦ Multi thread/Multi process
 - ♦ Vectorize operation (Like AVX)

Requirements - structure

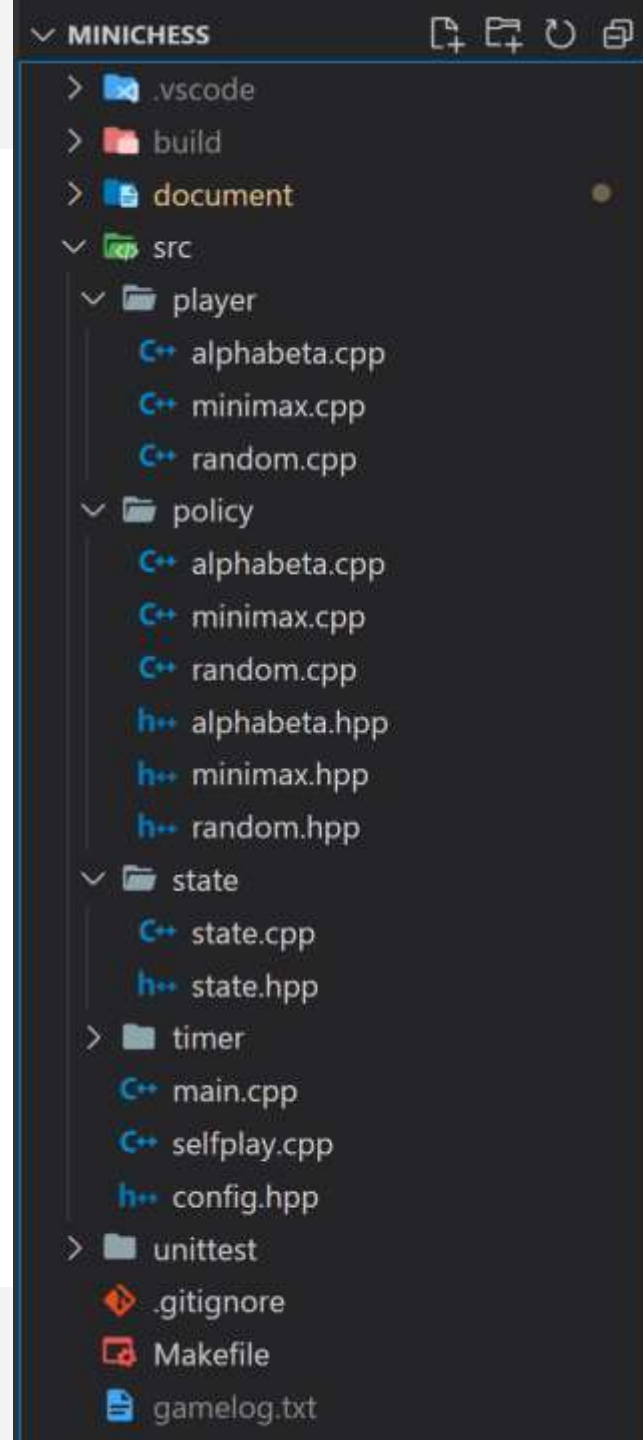
- Please use C++ and write your program with the structure in the folder (basically as same as right side, but only has random policy).
- Once you make player/xxx.cpp, policy/xxx.cpp, policy/xxx.hpp. You can use "make xxx" command to build the player with xxx policy. (For example, use "make alphabeta" with the structure on the right, will generate "build/player_alphabeta.exe file)

Make sure you have implemented all needed method.



Requirements - structure

- ♦ Make sure to **make a copy** of the player and policy you want to submit and rename them as “submission.cpp” for both player and policy.
 - ♦ For example, if I want to submit alphabeta player as my result. I should make a copy of player/alphabeta.cpp and a copy of policy/alphabeta.cpp. And then rename them to player/submission.cpp and policy/submission.cpp.
- ♦ You should make sure “make submission” command can work!



Requirements - structure

- ♦ Your program will be compiled in a GNU/Linux environment by:

```
kblueleaf@RiceShower: /mnt/c/Users/apoll/Desktop/code/project/MiniChess$ make
g++ --std=c++20 -Wall -Wextra -O3 -o build/main src/state/state.cpp src/main.cpp
src/main.cpp: In function 'void launch_executable(std::string)':
src/main.cpp:31:9: warning: ignoring return value of 'int system(const char*)' declared with attribute 'warn_unused_result' [-Wunused-result]
   31 |     system(command.c_str());
      |     ~~~~~^
src/main.cpp: In function 'int main(int, char**)':
src/main.cpp:92:11: warning: ignoring return value of 'int system(const char*)' declared with attribute 'warn_unused_result' [-Wunused-result]
   92 |     system("clear");
      |     ~~~~~^
g++ --std=c++20 -Wall -Wextra -O3 -o build/player_alphabeta src/state/state.cpp src/policy/alphabeta.cpp src/player/alphabeta.cpp
g++ --std=c++20 -Wall -Wextra -O3 -o build/player_minimax src/state/state.cpp src/policy/minimax.cpp src/player/minimax.cpp
g++ --std=c++20 -Wall -Wextra -O3 -o build/player_random src/state/state.cpp src/policy/random.cpp src/player/random.cpp
```

- ♦ Please make sure your program can be compiled by the command above with no error.
(If you don't change the makefile, just use "make" command and see if there is any error.)

Requirements - Report and Demo

- ♦ You should write a report to elaborate on how you design your AI
- ♦ If you have implemented NNUE or MCTS, please provide a brief description in your report of how it was implemented, along with code explanations if possible.
- ♦ 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. Chess and Mini Chess
3. State Value Function
4. Minimax
5. Alpha-Beta Pruning
6. How To Design Your AI
7. Package
8. Requirements
9. Grading
10. Submission

Grading

- ♦ The project accounts for 9 points of your total grade
- ♦ Beat every baseline => +5 points
(1 point for each 4 baselines, 1 point if you beat all the baselines with both white and black)
- ♦ Implement Tree search (Minimax) => +2 points
- ♦ Design of your state value function => +1 point
- ♦ Implement Alpha-Beta Pruning => +1 point

Grading - baselines

- ♦ We have 4 baselines:
 - ♦ Random
 - ♦ Weak MiniMax
 - ♦ Strong MiniMax
 - ♦ Strong AlphaBeta
- ♦ Your program will play with baselines for both white and black.
- ♦ If you can get 1win + 1draw (or 2win), you can go to next baselines and get the score.
- ♦ If you can beat all the baselines with 8wins, you get final 1 point.

Grading Bonus

- ♦ (Bonus) Use version control software => +1 point
 - ♦ You need to push your code to github as submission. But if you also use github+git as version control software and have at least 3 commits, you will get this point.
- ♦ (Bonus) Class ranking => At most +2 points
 - ♦ You can attend the class ranking if you beat all baselines (8wins).
 - ♦ Your AI will play against other AIs of your classmates and gain a bonus score according to your ranking.
- ♦ (Bonus) Advanced algorithm => MCTS: +1points, NNUE: +2points
 - ♦ You get this bonus only if your advanced algorithm can beat the first 3 baselines and you can explain it well.

Grading Plagiarism

- ♦ We will compare the code from both classes and previous years to detect plagiarism.
- ♦ **Please set your GitHub repository to private before the deadline** to prevent your code from being plagiarized by others. Remember to reset it for the public after the deadline.
- ♦ If plagiarism is still found, we will determine it based on the chronological order of commits on GitHub. **The plagiarizer will receive 0 points, while the plagiarized person will receive a 50% deduction in score.**

Outline

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

Submission

- Submit the report to Mini Project 3 繳交區 on eeclass
File name: <student_id>_project3.pdf
Ex: 111000000_project3.pdf
- Complete the following Google Form, which includes your GitHub Repo link and check of your bonus implementation
- Link: <https://forms.gle/V9CAMh3yMychxW8z7>
- **Deadline: 6/20** (both report and google form)
- **Late submissions within 2 hours will receive a 40% deduction in score.(六折)**
- **Late submissions more than 2 hours will receive a score of 0.**

Happy Coding!