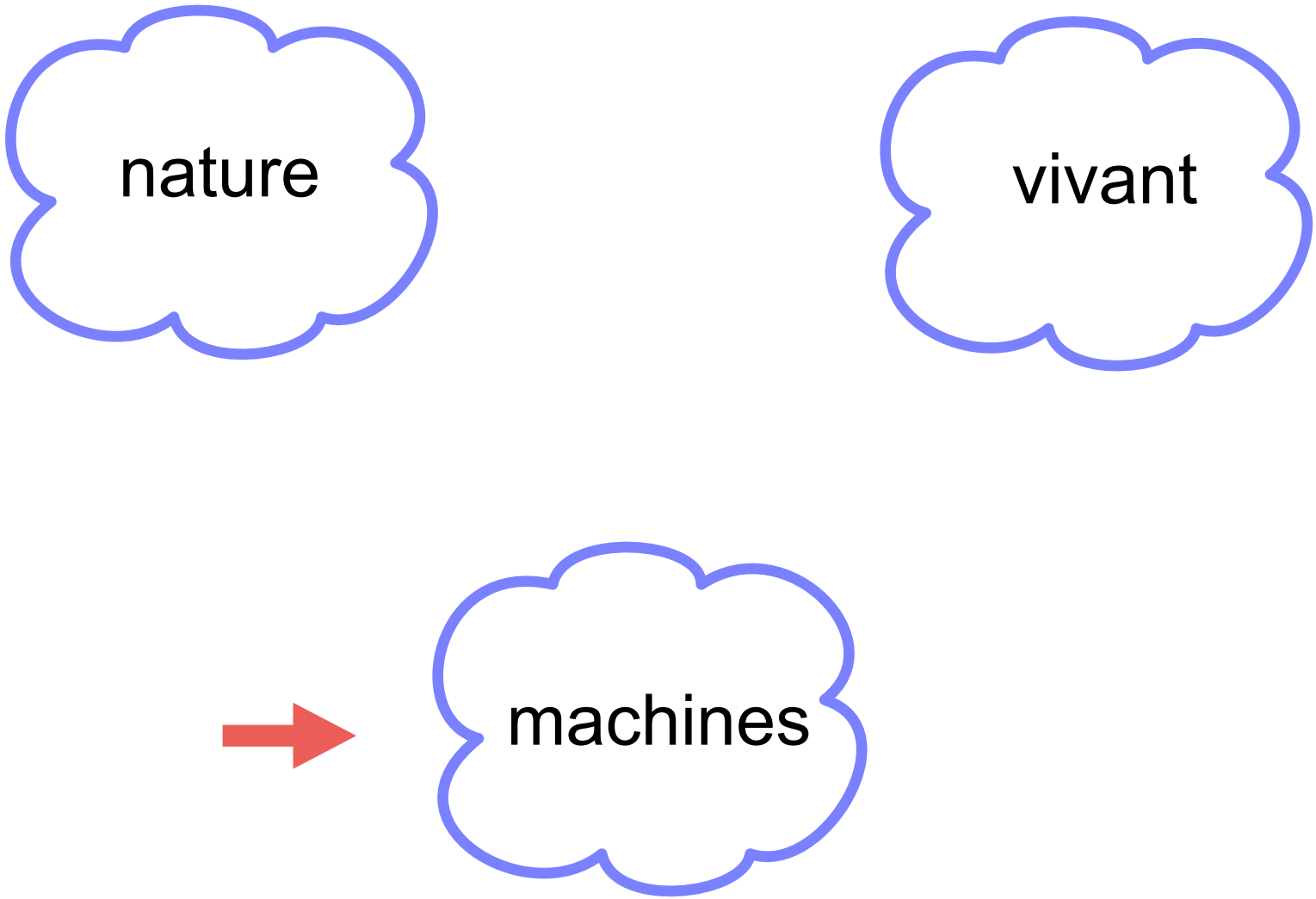

Informatique
Jean-Jacques Lévy
INRIA

La Science

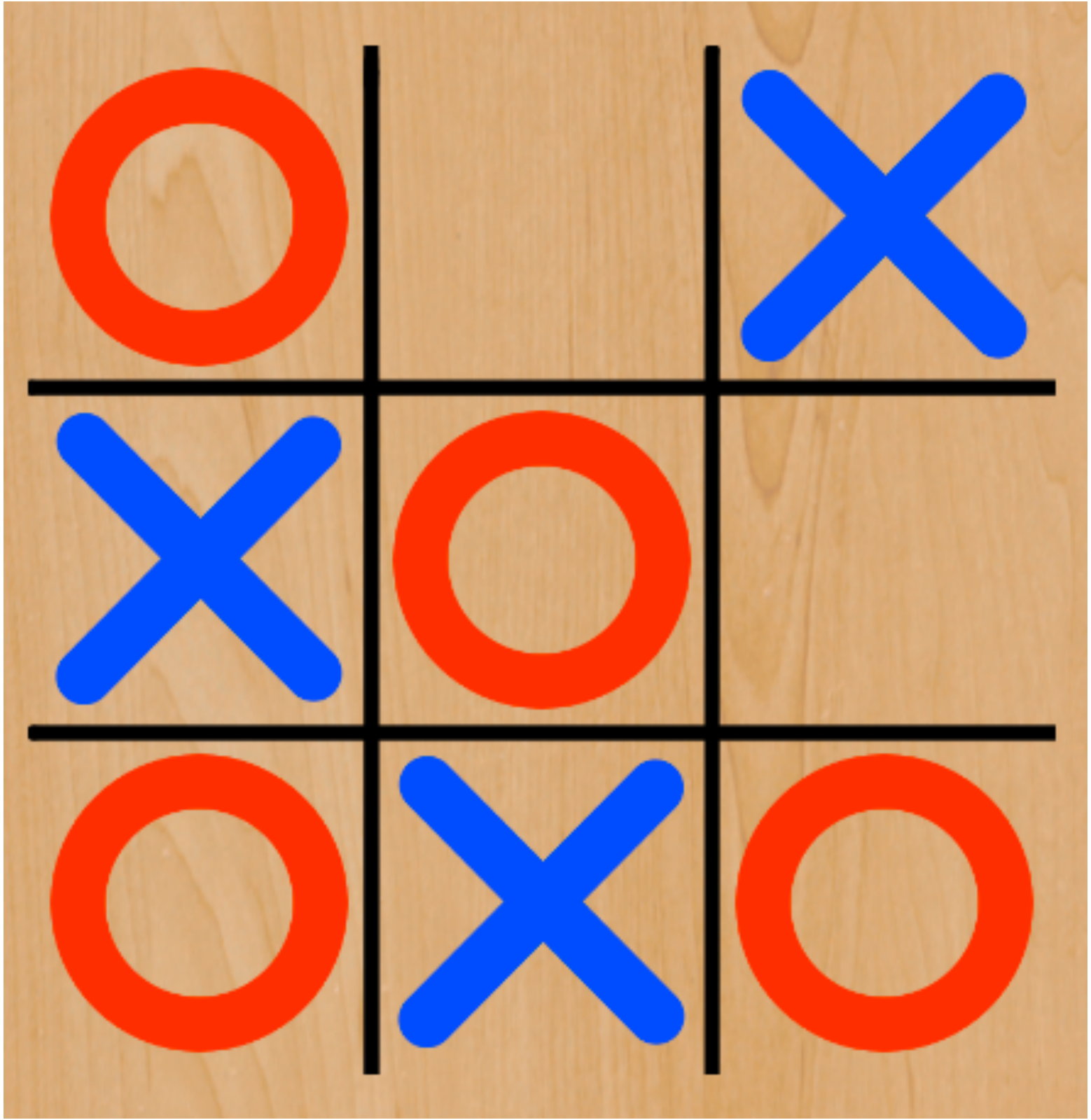


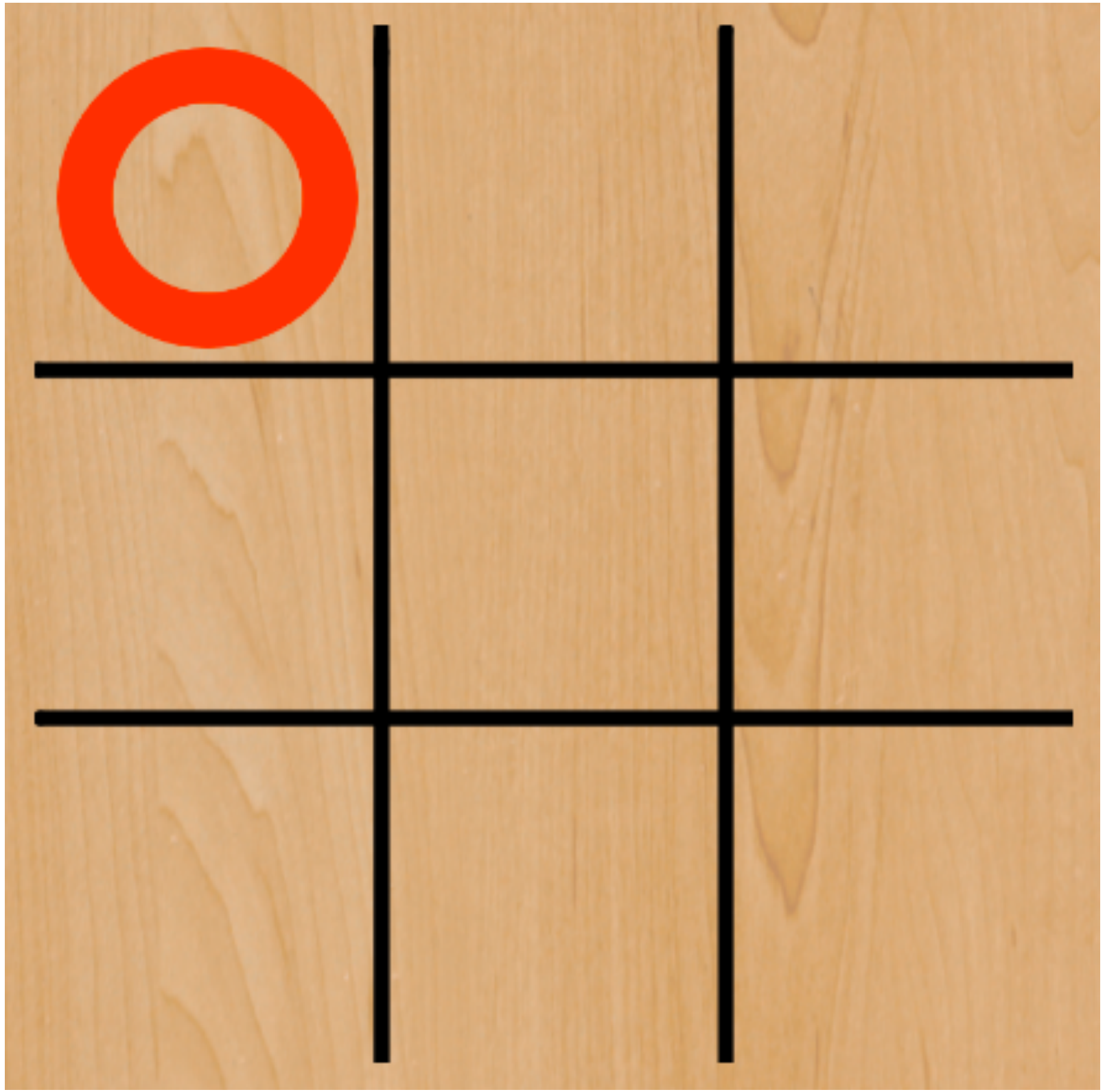
DES

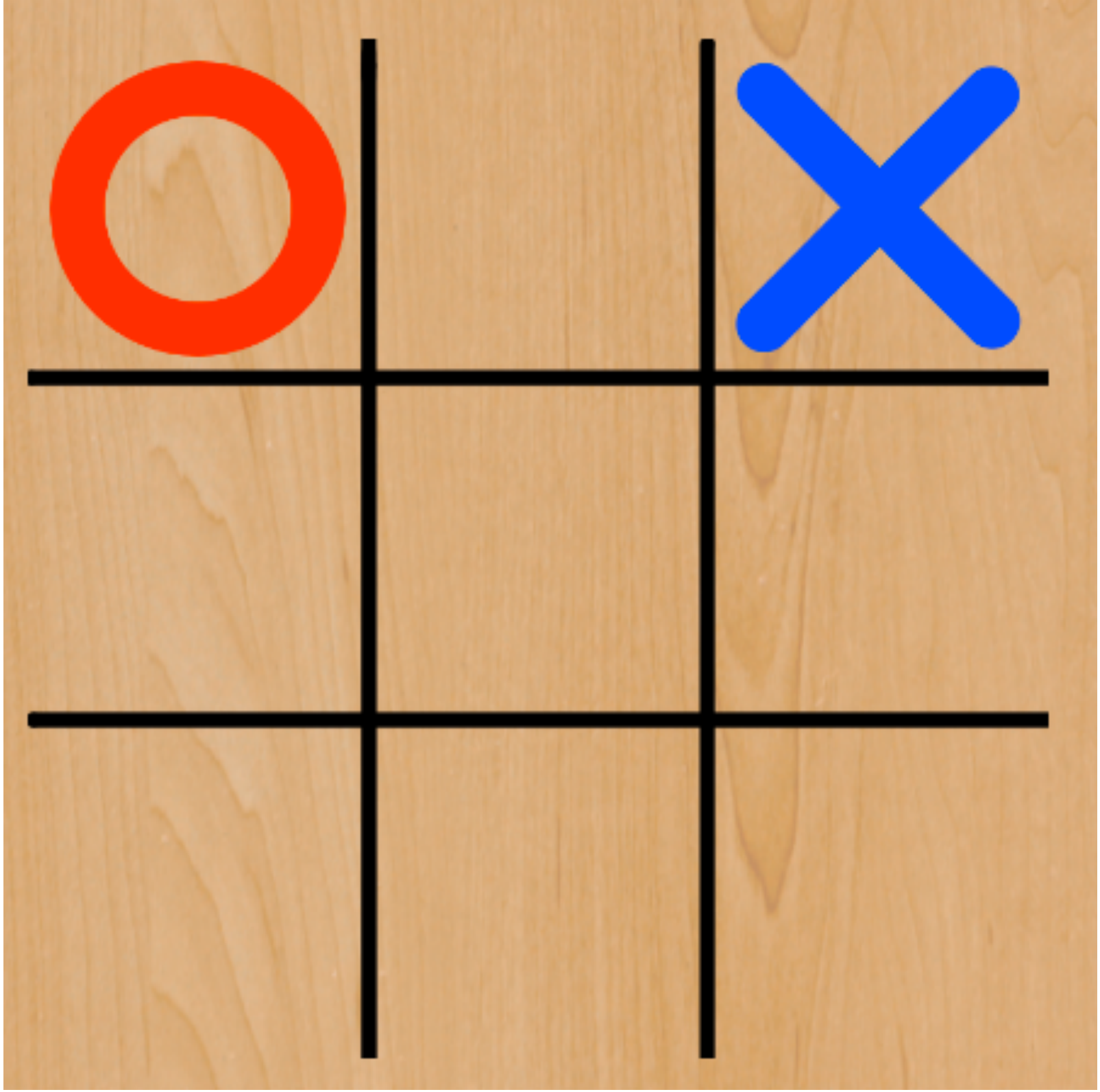
MACHINES

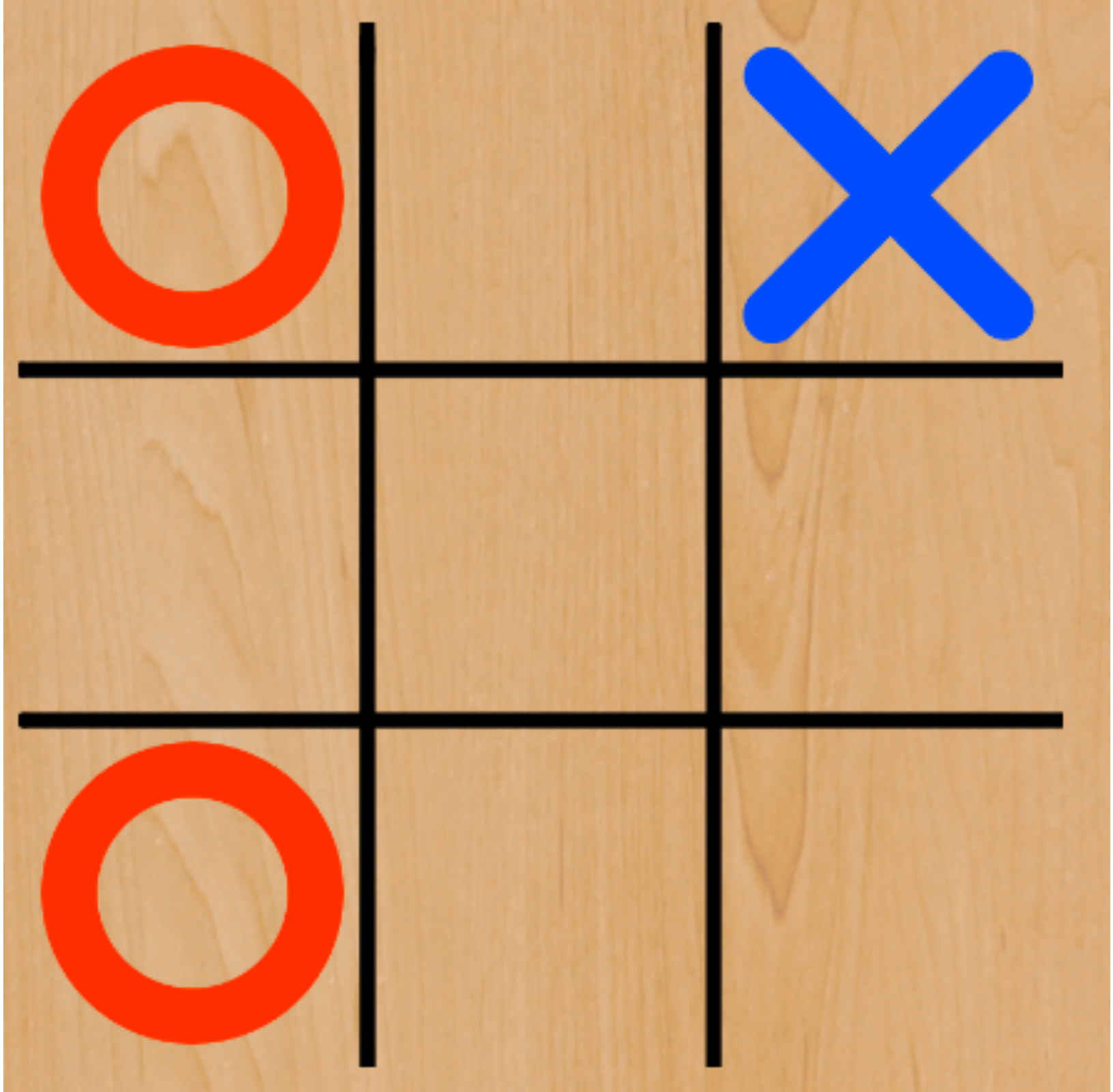
(1945)

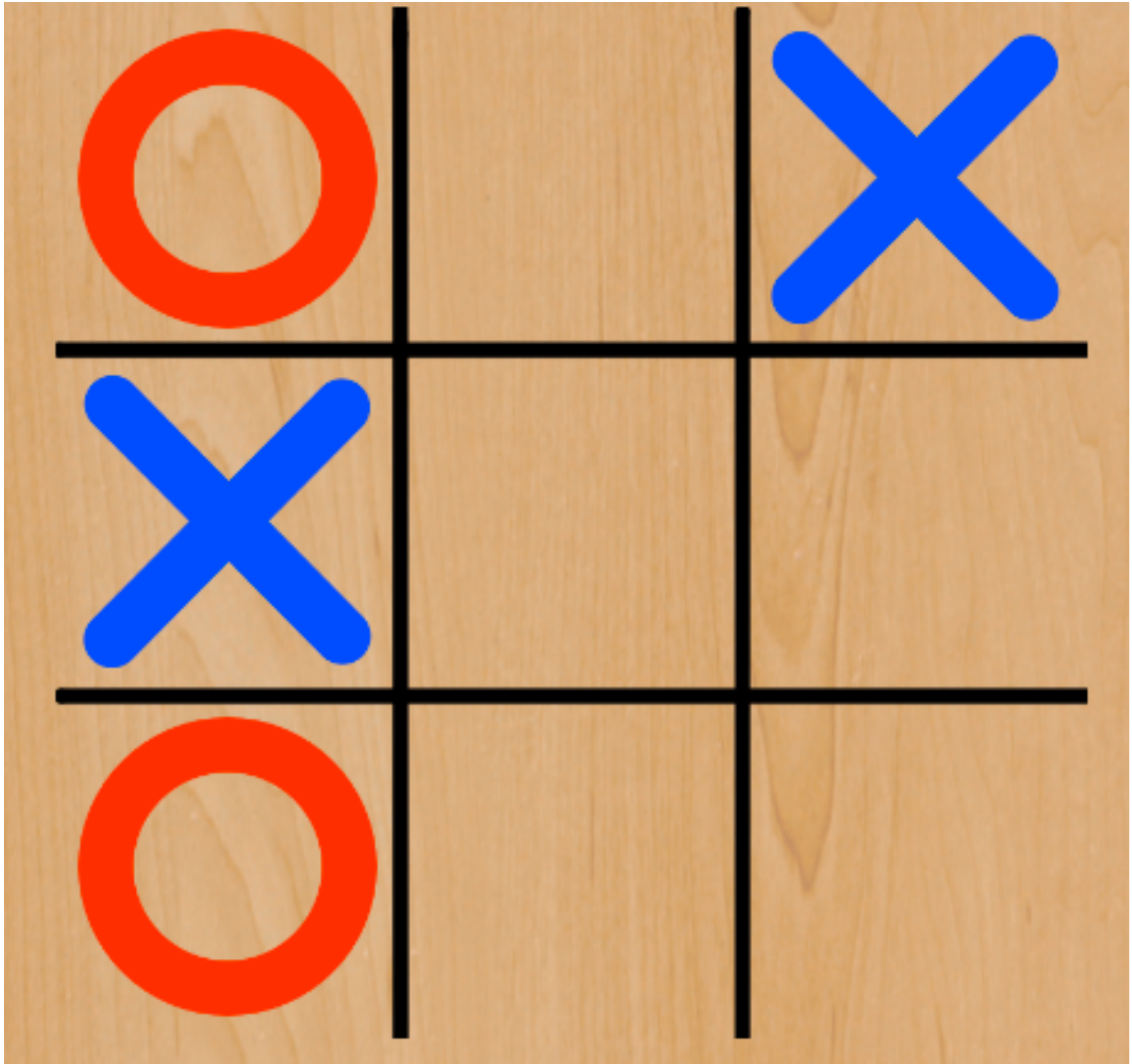
PROGRAMMES

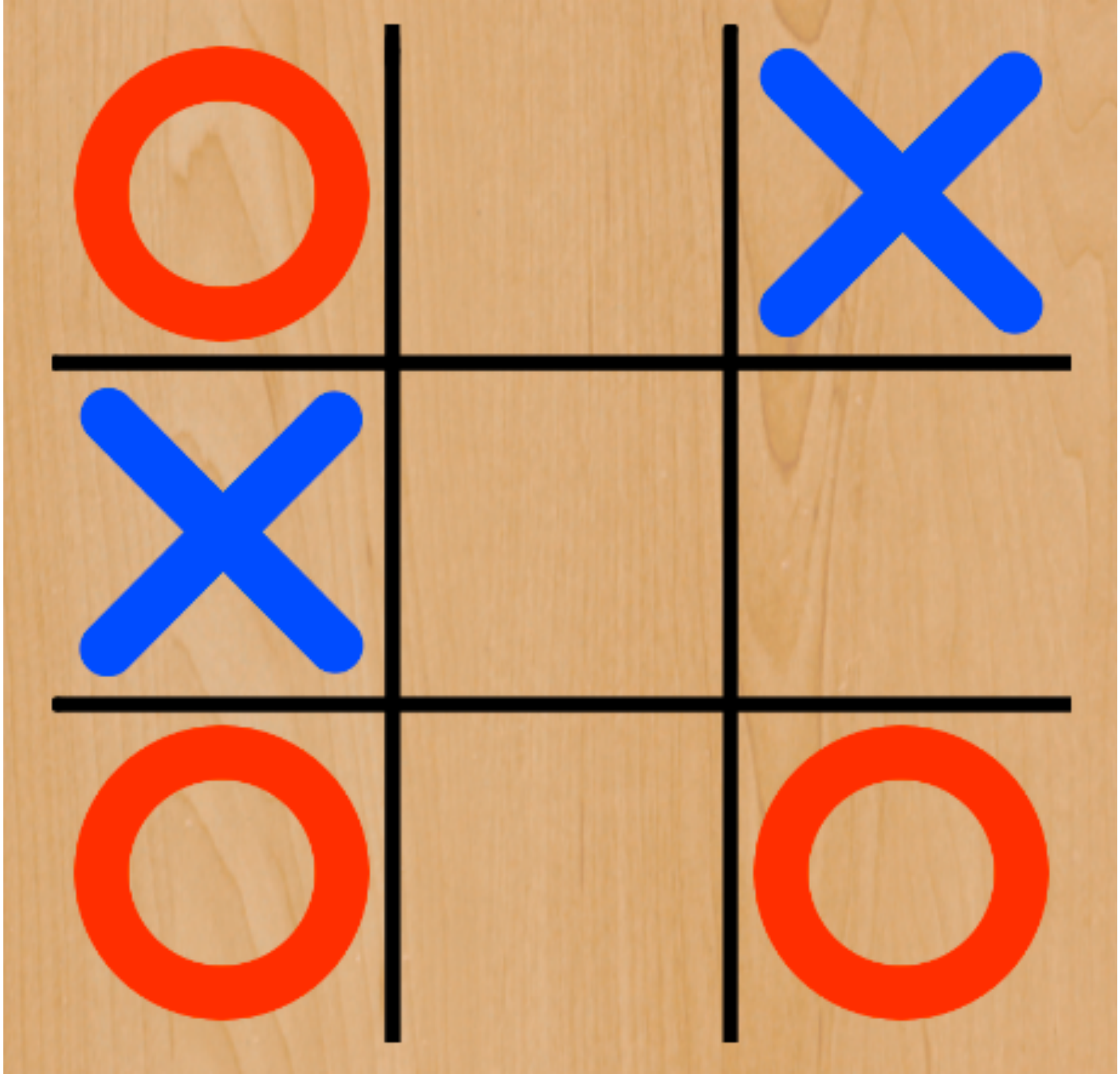


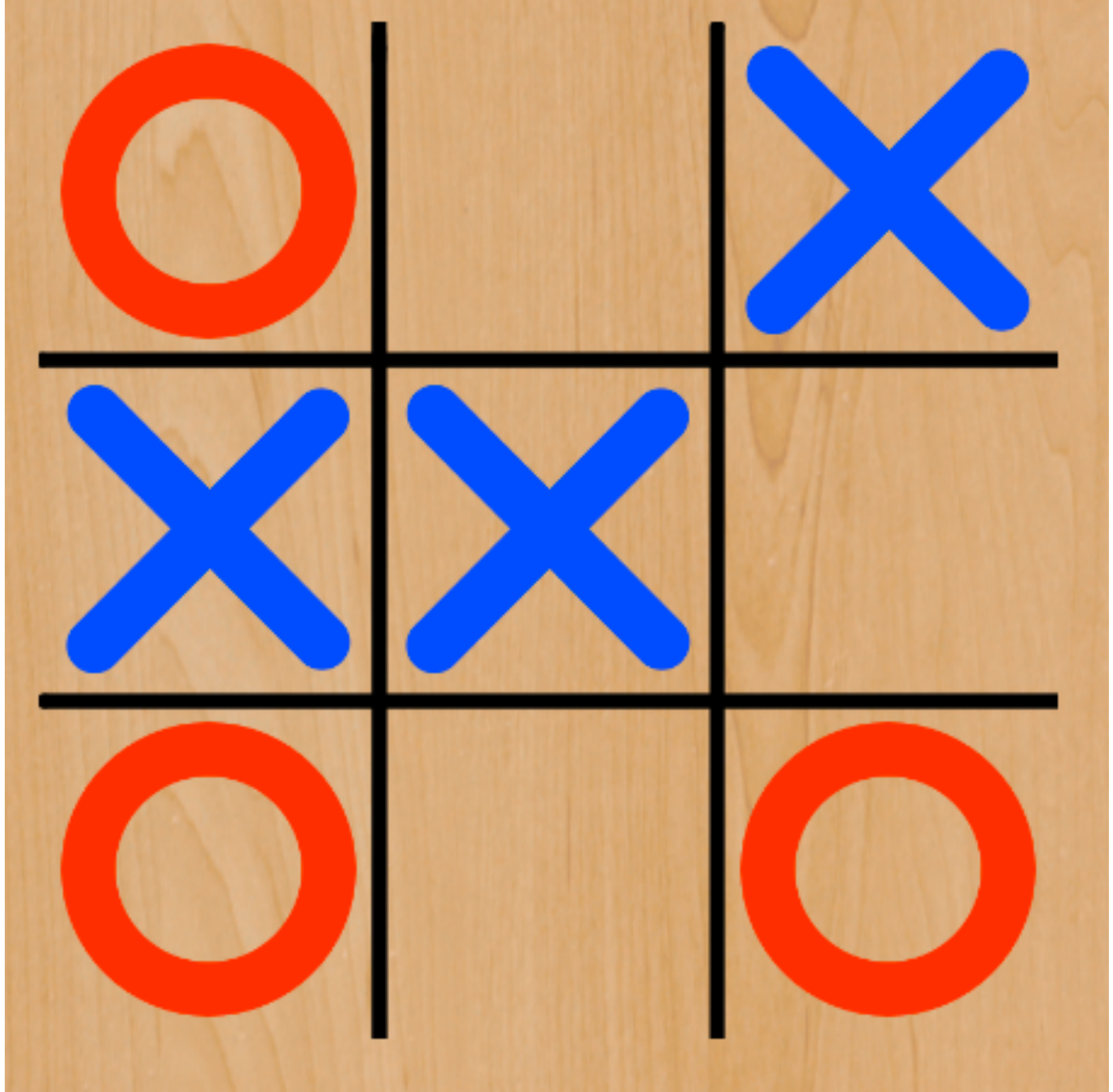


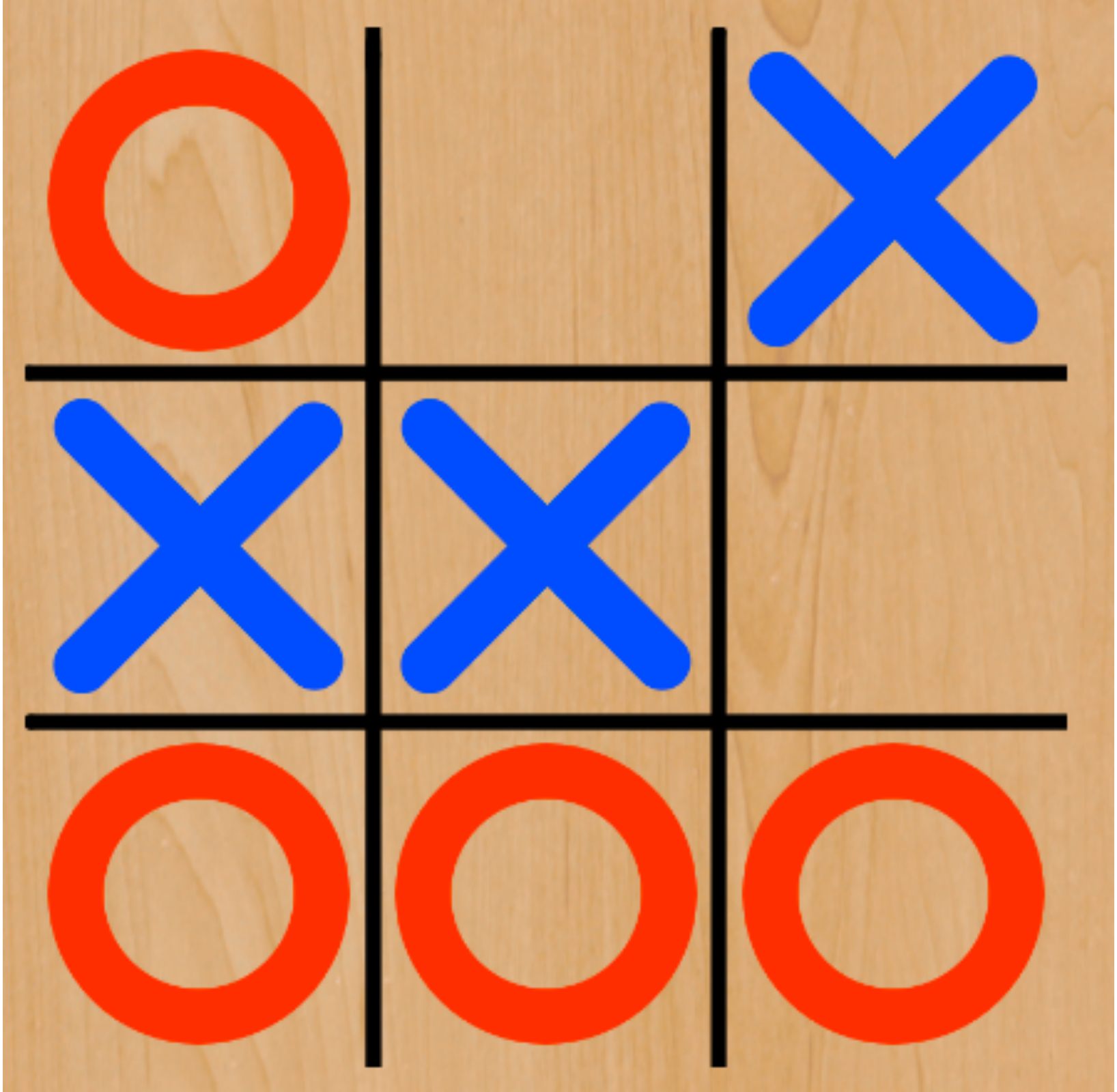












```
type brick = Empty | Computer | User

let brickToChar x = match x with
 | Empty -> ' '
 | Computer -> 'X'
 | User -> 'O' ;;

let emptyBoard = Array.make_matrix 3 3 Empty ;;

let copyBoard b = let b' = Array.make_matrix 3 3 Empty in
  for i=0 to 2 do for j=0 to 2 do b'.(i).(j) <- b.(i).(j) done done;
  b';;

let get b (i,j) = b.(i).(j) ;;
let set player b (i,j) =
  let b' = copyBoard b in b'.(i).(j) <- player; b' ;;

let winningPositions = [
 [(0,0) ; (1,1); (2,2)];
 [(2,0) ; (1,1); (0,2)];
 [(0,0) ; (0,1); (0,2)];
 [(1,0) ; (1,1); (1,2)];
 [(2,0) ; (2,1); (2,2)];
 [(0,0) ; (1,0); (2,0)];
 [(0,1) ; (1,1); (2,1)];
 [(0,2) ; (1,2); (2,2)]
];;

let allPositions = [
 (0,0); (0,1); (0,2);
 (1,0); (1,1); (1,2);
 (2,0); (2,1); (2,2) ] ;;

let hasWon player board =
  List.exists (function ligne ->
      List.for_all (function (i,j) -> player = board.(i).(j)) ligne)
   winningPositions ;;
```

```
let freeSpace board =
  List.filter (function (i,j) -> board.(i).(j) = Empty) allPositions

(******************** strategy *******************************)
type evaluation =  Win | Draw | Lose

let rec evaluate board move =
  let b2 = set Computer board move in
  if hasWon Computer b2 then Win
  else
    match freeSpace b2 with
    | [ ] -> Draw
    | userChoices ->
       let b3s = List.map (set User b2) userChoices in
       if List.exists (hasWon User) b3s then Lose
       else if List.exists (fun b3 -> bestOutcome b3 = Lose) b3s
       then Lose
       else if List.exists (fun b3 -> bestOutcome b3 = Draw) b3s
       then Draw
       else Win

and findBestChoice b =
  match freeSpace b with
  | [ ] -> ((-1,-1), Draw)
  | choices ->
    try let c = List.find (fun c -> evaluate b c = Win) choices in
      (c, Win)
    with Not_found ->
    try let c = List.find (fun c -> evaluate b c = Draw) choices in
      (c, Draw)
    with Not_found ->
      (List.hd choices, Lose)

and bestOutcome b = snd (findBestChoice b)
```

```
let bestChoice b = fst (findBestChoice b)

(******************** interface ********************************)
let computerPlay b = set Computer b (bestChoice b)

let printBoard b =
  Printf.printf "   | A | B | C |\n" ;
  Printf.printf "---+---+---+---+\n" ;
  for y=0 to 2 do
   Printf.printf " %d | %c | %c | %c | \n" (y + 1)
    (brickToChar (get b (y,0)))
    (brickToChar (get b (y,1)))
    (brickToChar (get b (y,2)))
  done;
  Printf.printf "---+---+---+---+ \n" ;;

let rec userPlay b =
  Printf.printf "Quelle case jouez-vous? (format: a1)\n" ;
  let input = read_line() in
  if String.length input <> 2
     || input.[0] < 'a' || input.[0] > 'c'
     || input.[1] < '1' || input.[1] > '3' then begin
     Printf.printf "N'importe quoi! \n";
     userPlay b
  end else
     let y = int_of_char(input.[0]) - int_of_char('a')  in
     let x = int_of_char (input.[1]) - int_of_char ('1') in
     if get b (x,y) <> Empty then begin
       Printf.printf "Case pas libre.\n";
       userPlay b
     end else
       set User b (x,y) ;;
```

```
let rec gameLoop b player =
  if freeSpace b = [ ] then
    Printf.printf "Fin. Match nul \n"
  else if player = Computer then begin
    Printf.printf "Ordinateur joue... \n";
    let b2 = computerPlay b in
    printBoard b2;
    if hasWon Computer b2 then
      Printf.printf "Fin. J'ai gagné \n"
    else
      gameLoop b2 User
  end else if player = User then
    let b2 = userPlay b in
    printBoard b2;
    if hasWon User b2 then
      Printf.printf "Fin. Vous avez gagné \n"
    else
      gameLoop b2 Computer
   ;;

let choose positions =
  let n = List.length positions in
  List.nth positions (Random.int n) ;;

let rec beginner () =
  let input = read_line() in
  if String.length input <> 1 then beginner() else
  if input.[0] = 'O' || input.[0] = 'o' || input.[0] = '0' then
    User
  else if input.[0] = 'X' || input.[0] = 'x' then
    Computer
  else beginner() ;;
```

```
let main () =
  Random.self_init() ;
  Printf.printf "Qui démarre? O/X :: " ;
  let player = beginner() in
  if player = Computer then
    let b = set Computer emptyBoard (choose (freeSpace emptyBoard)) in
    printBoard b ;
    gameLoop b User
  else
    gameLoop emptyBoard User;;


main();;
```

Les programmes

- on apprend des programmes à une machine
- et elle exécute les programmes
- elle va très vite (10 000 000 opérations par seconde)
- la machine ne se fatigue jamais
- mais elle apprend lentement
  (écrire des programmes prend beaucoup de temps)

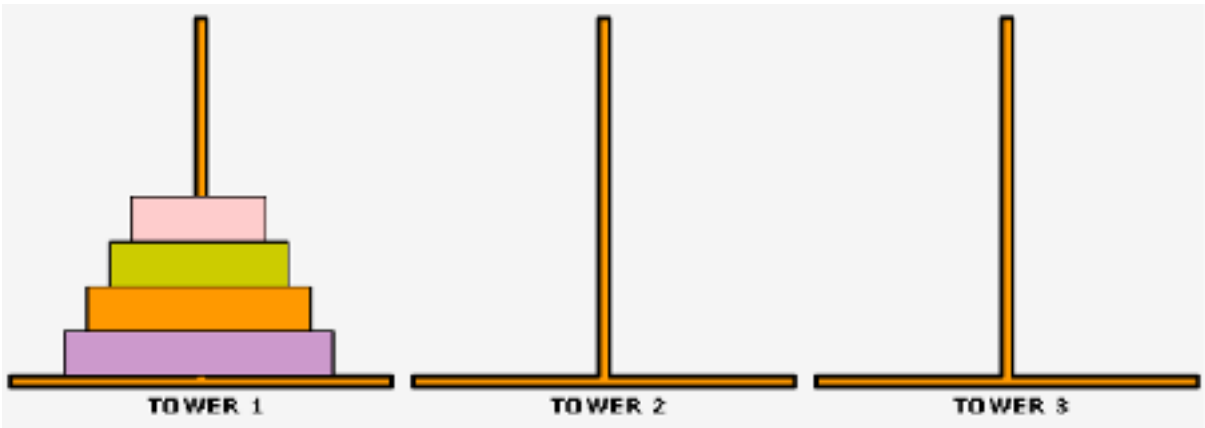
Opérations élémentaires

- additions, soustractions, multiplications
- démonstration avec Maple

TOURS DE HANOI

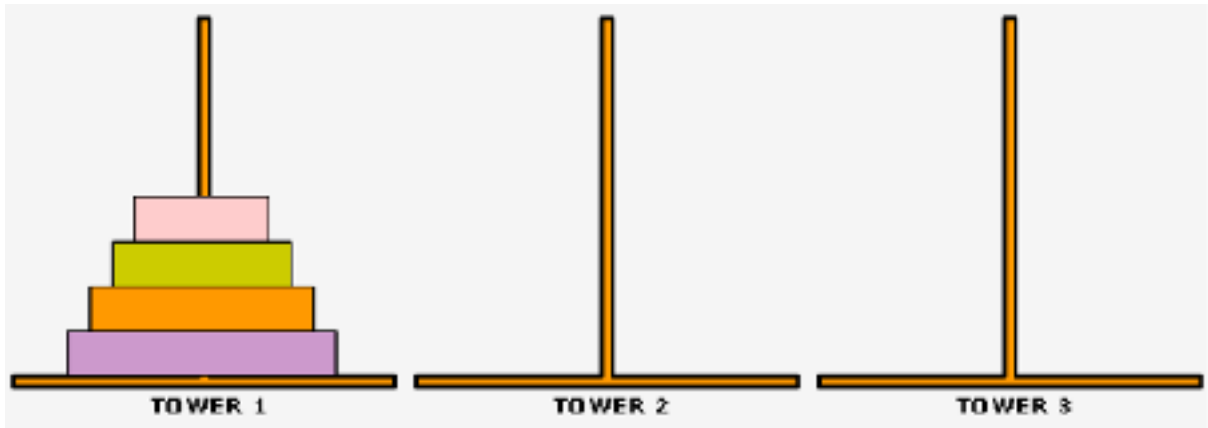
(récursivité)

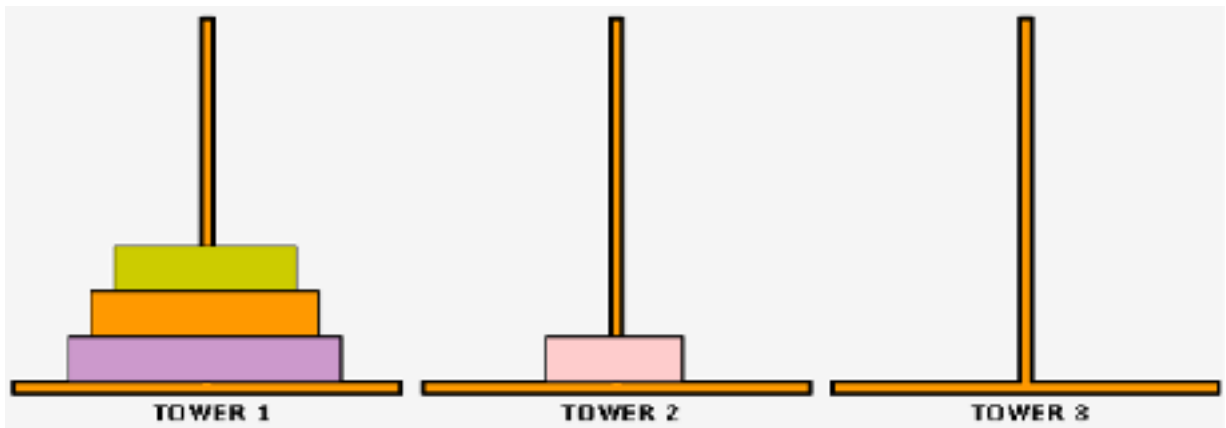
Tours de Hanoi



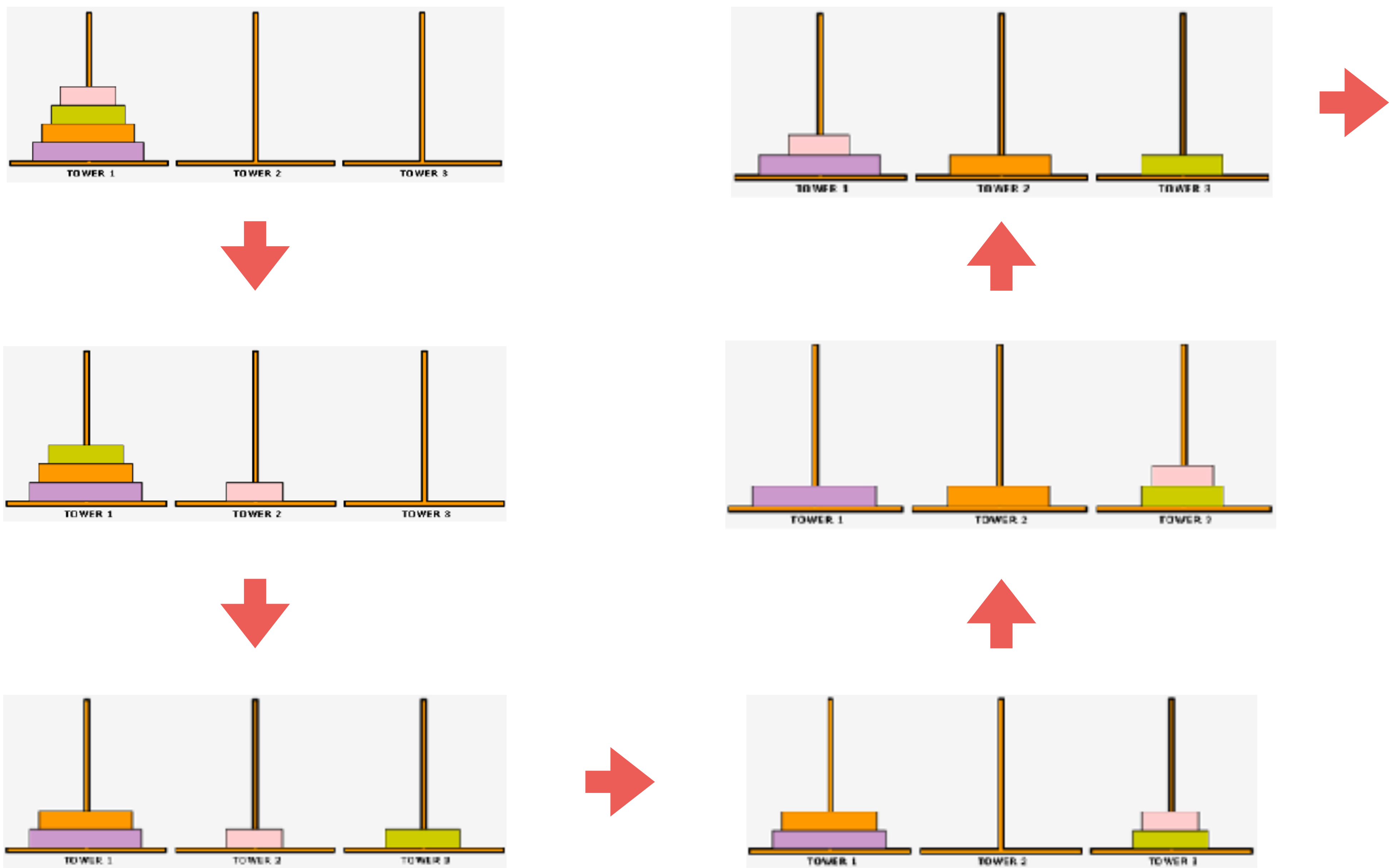
jamais petit
sous un grand

Tours de Hanoi

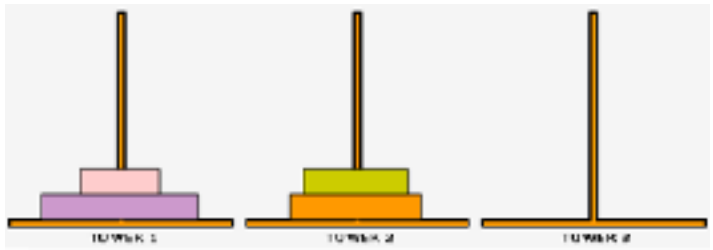


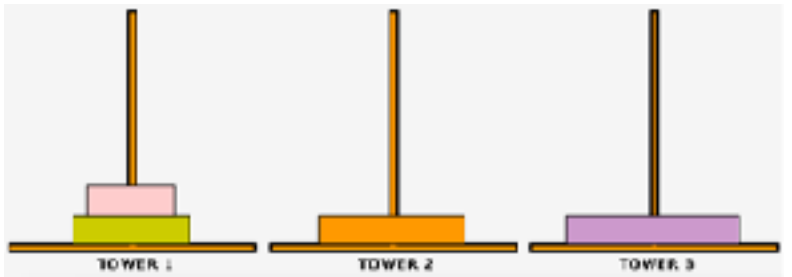


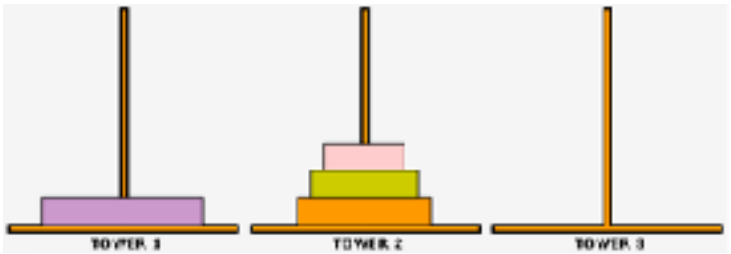
Tours de Hanoi

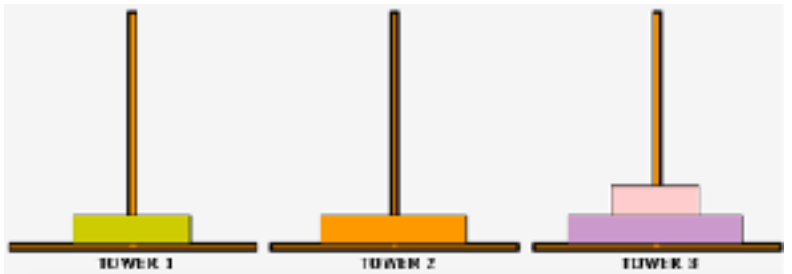


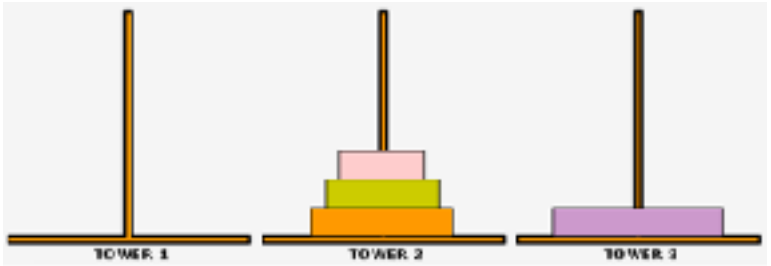
Tours de Hanoi

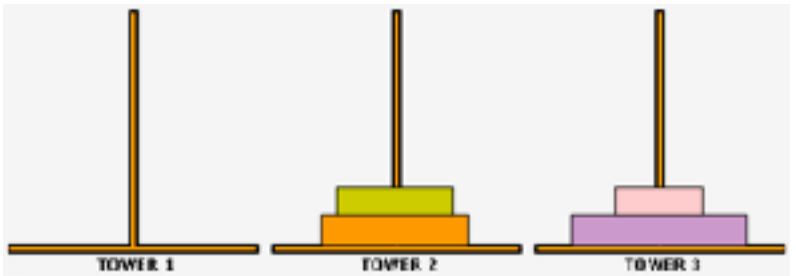




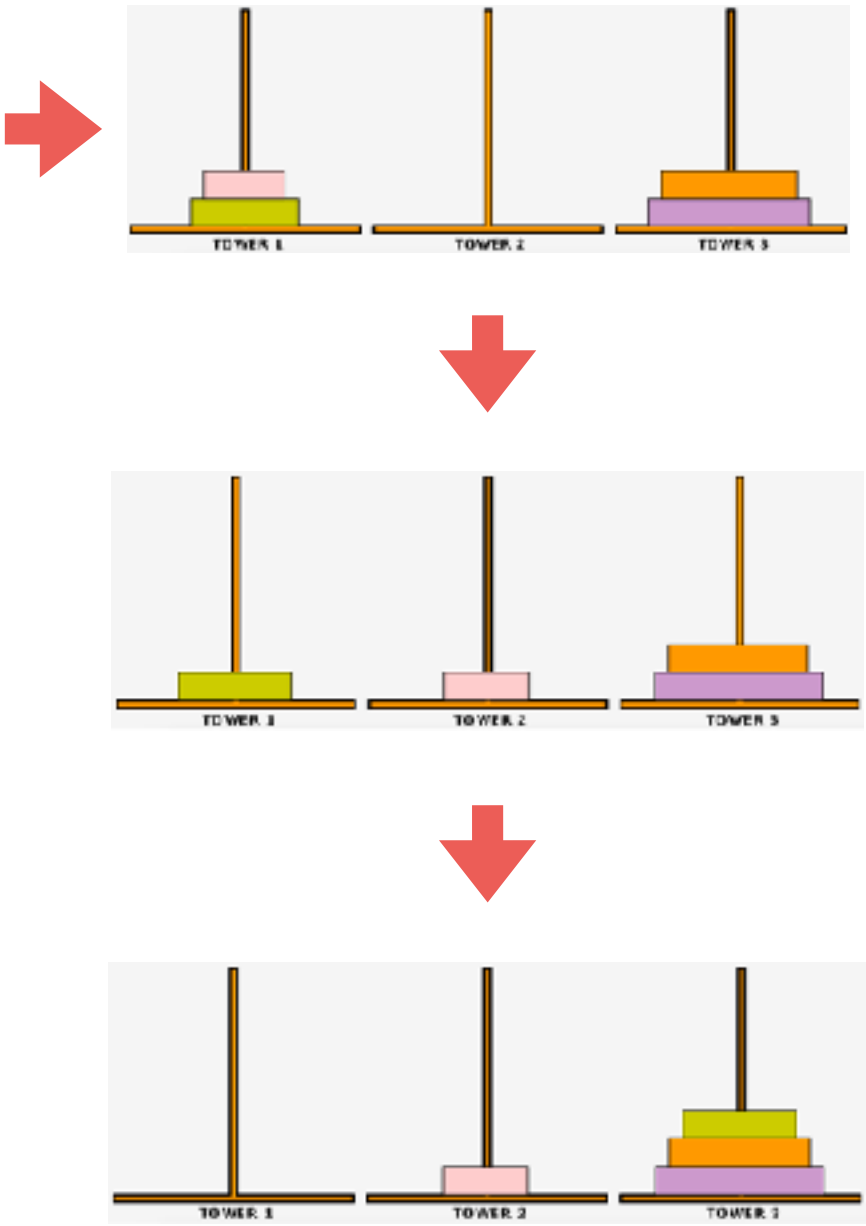




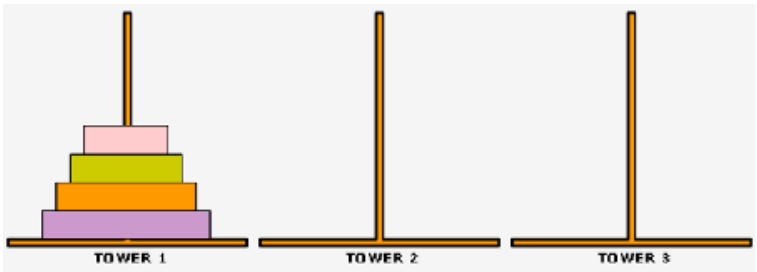


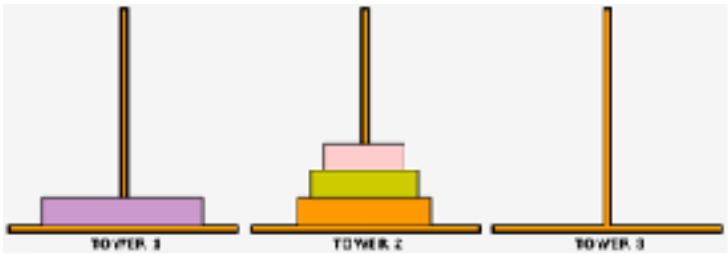


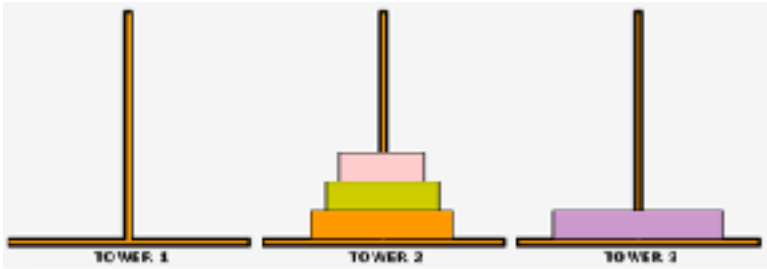
Tours de Hanoi



Tours de Hanoi



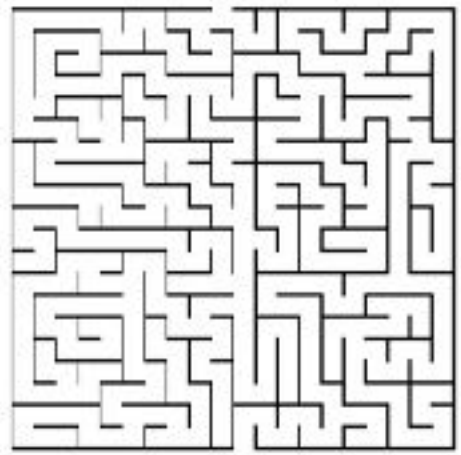


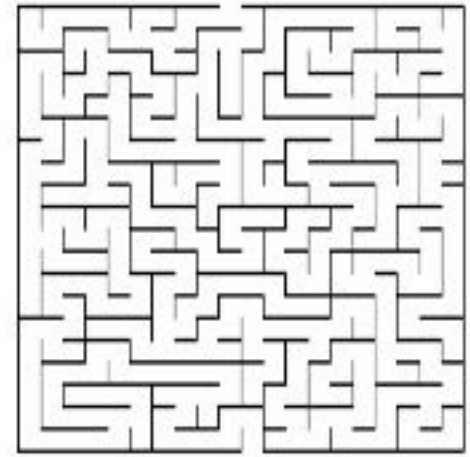


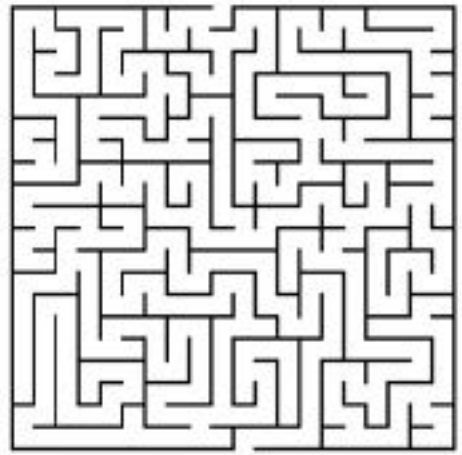

```
let tour_milieu i j =   6 - i -j ;;

let deplacer i j = Printf.printf "%d  -->  %d \n" i j ;;

let rec hanoi n i j =
  if n > 0 then
    let m = tour_milieu i j in
    hanoi (n - 1) i m;
    deplacer i j;
    hanoi (n - 1) m j;
  ;;

hanoi 4 1 3 ;;
```

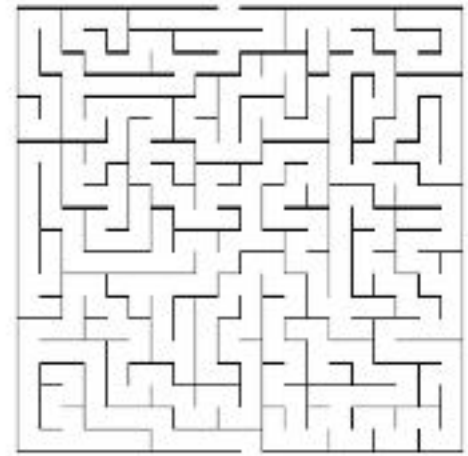
LABYRINTHES

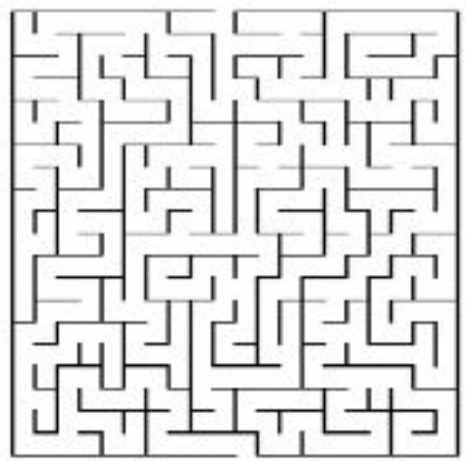
(recherche exhaustive)

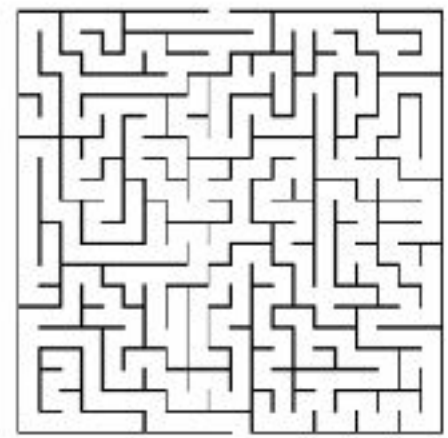


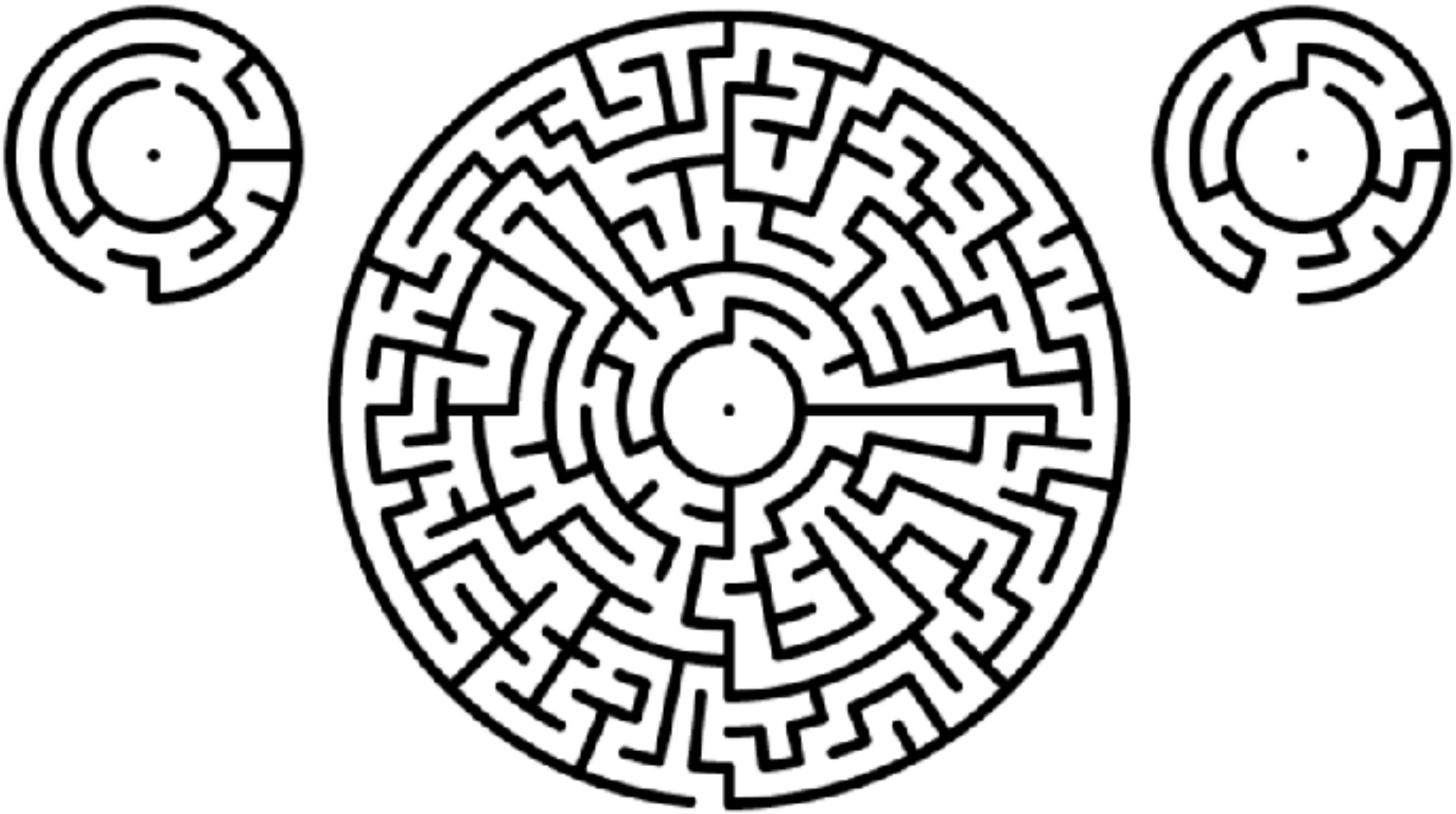


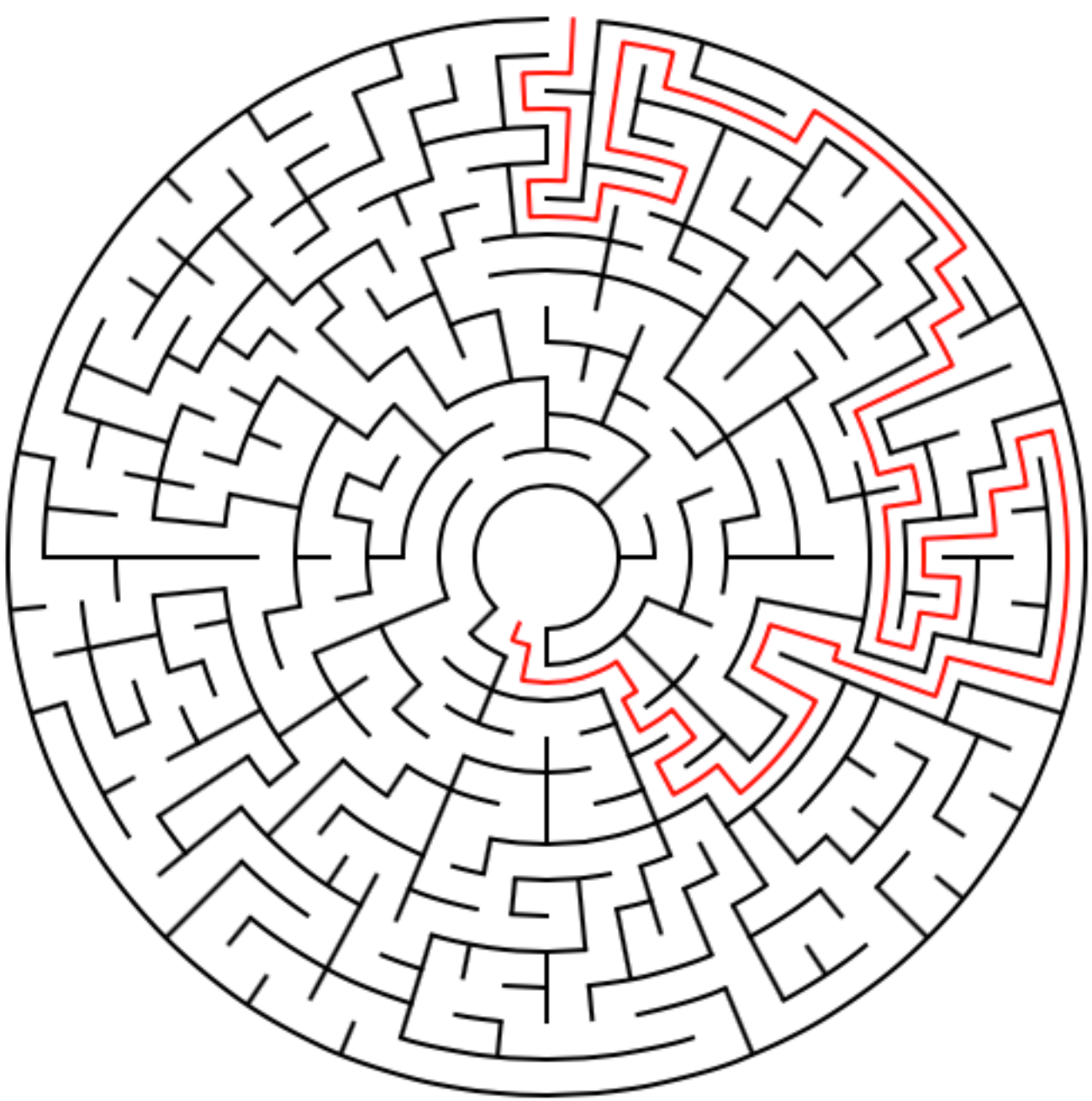












Sortie de labyrinthe

- technique 1: suivre le mur de droite

  (marche toujours ?)

- technique 2: le petit Poucet

  - on prend un chemin au hasard,

  - en laissant des petits cailloux pour ne pas repasser 2 fois par le même chemin

[en informatique: exploration avec backtracking]

LA MARCHE DU CAVALIER

(algorithme glouton)