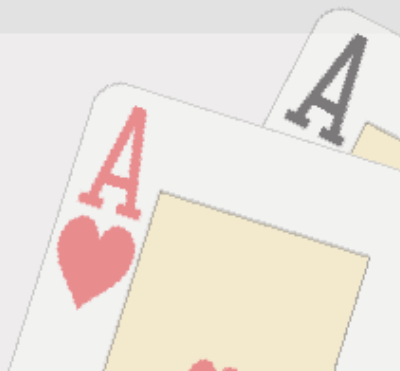


# AI Games course

Certificate 1, session 2

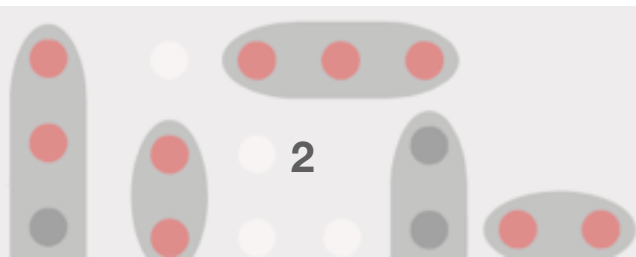


AIgaming.com



# The maze example

in	1	2	3	
	4	5	6	
	7	8	9	out



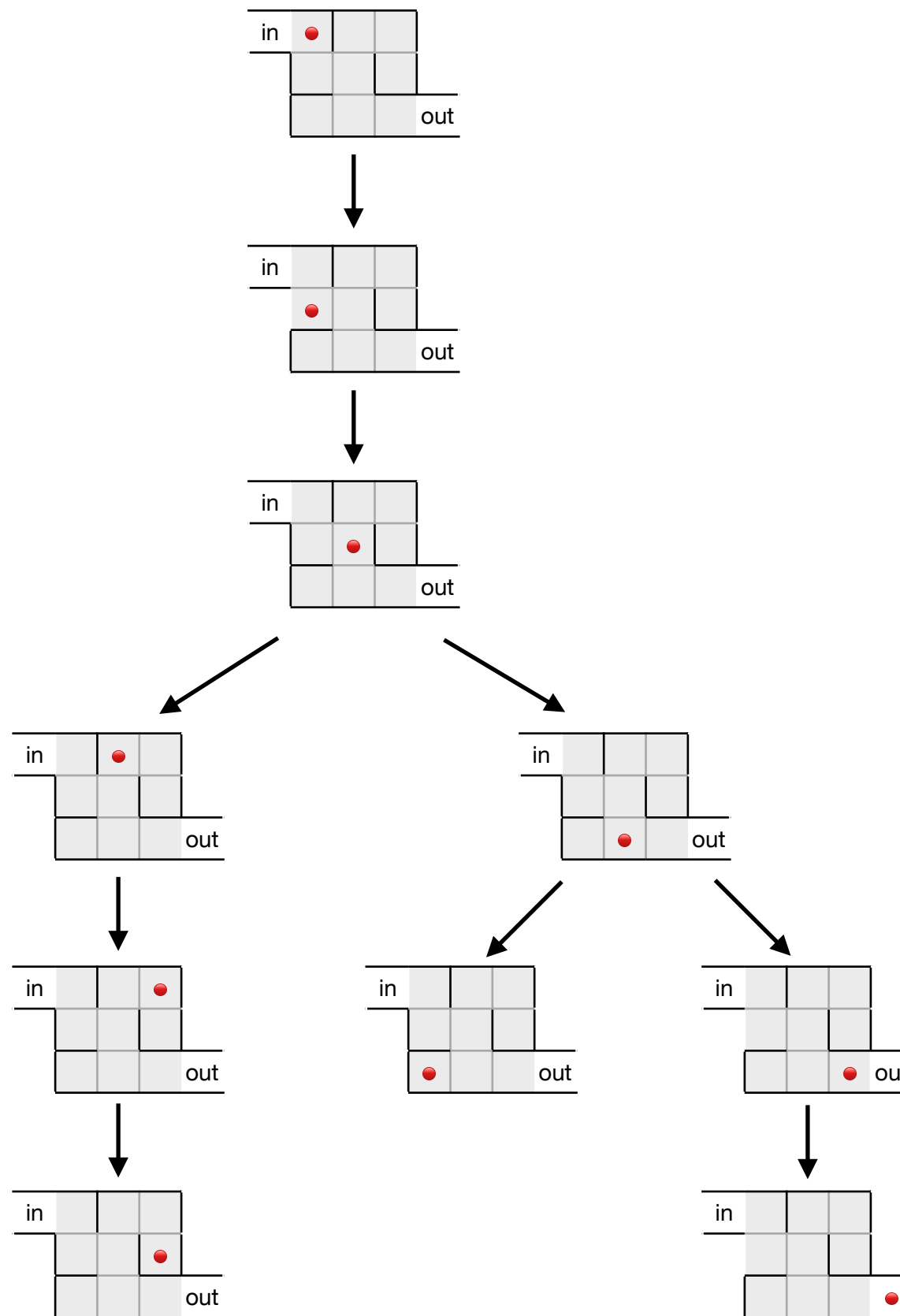
# The maze example: graph representation

in	1	2	3	
	4	5	6	
	7	8	9	out

```
maze = {'in': set([1]),  
        1: set(['in', 4]),  
        2: set([3, 5]),  
        3: set([2, 6]),  
        4: set([1, 5]),  
        5: set([2, 4, 8]),  
        6: set([3]),  
        7: set([8]),  
        8: set([5, 7, 9]),  
        9: set([8, 'out']),  
        'out': set([9])}
```

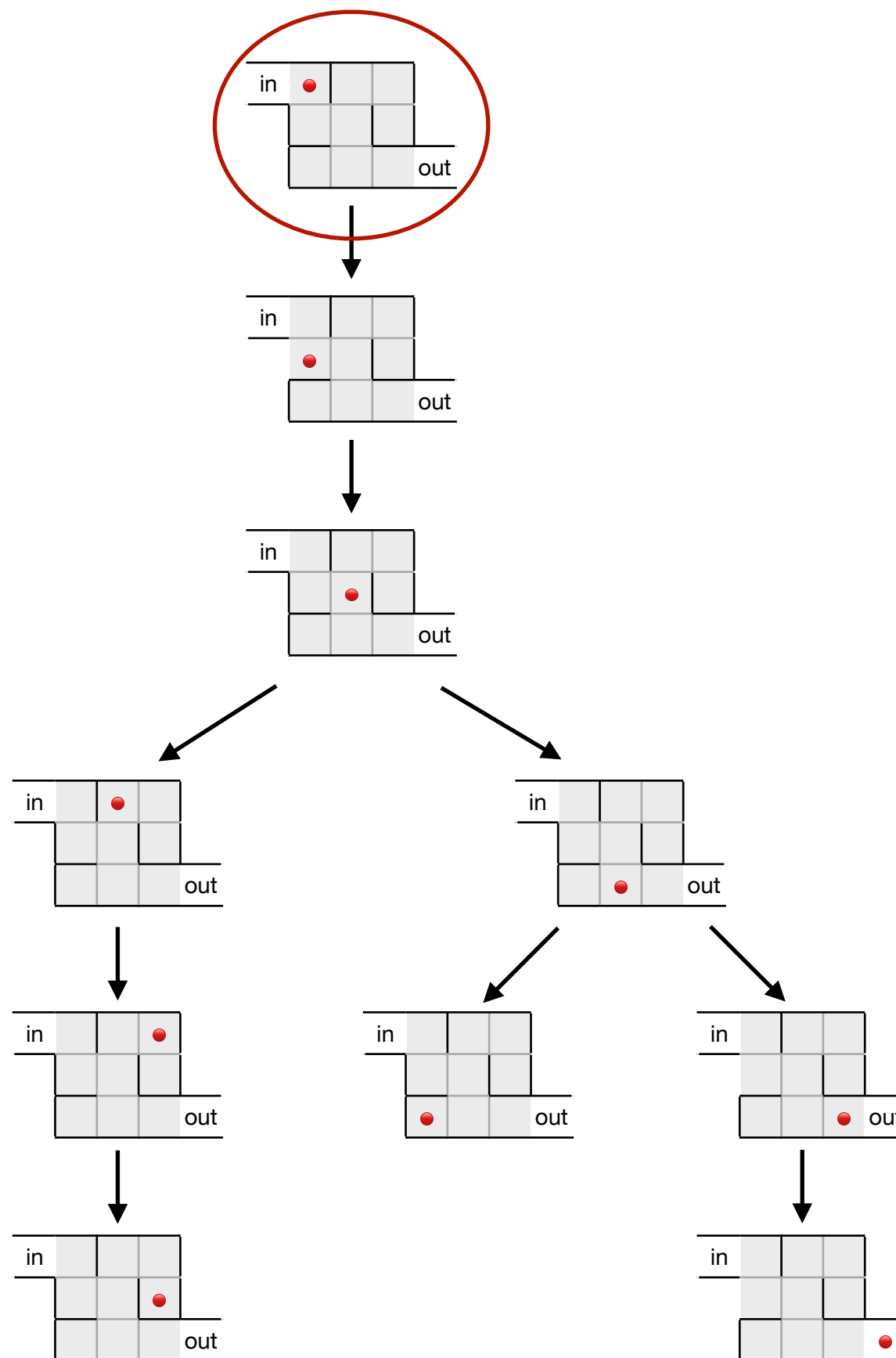


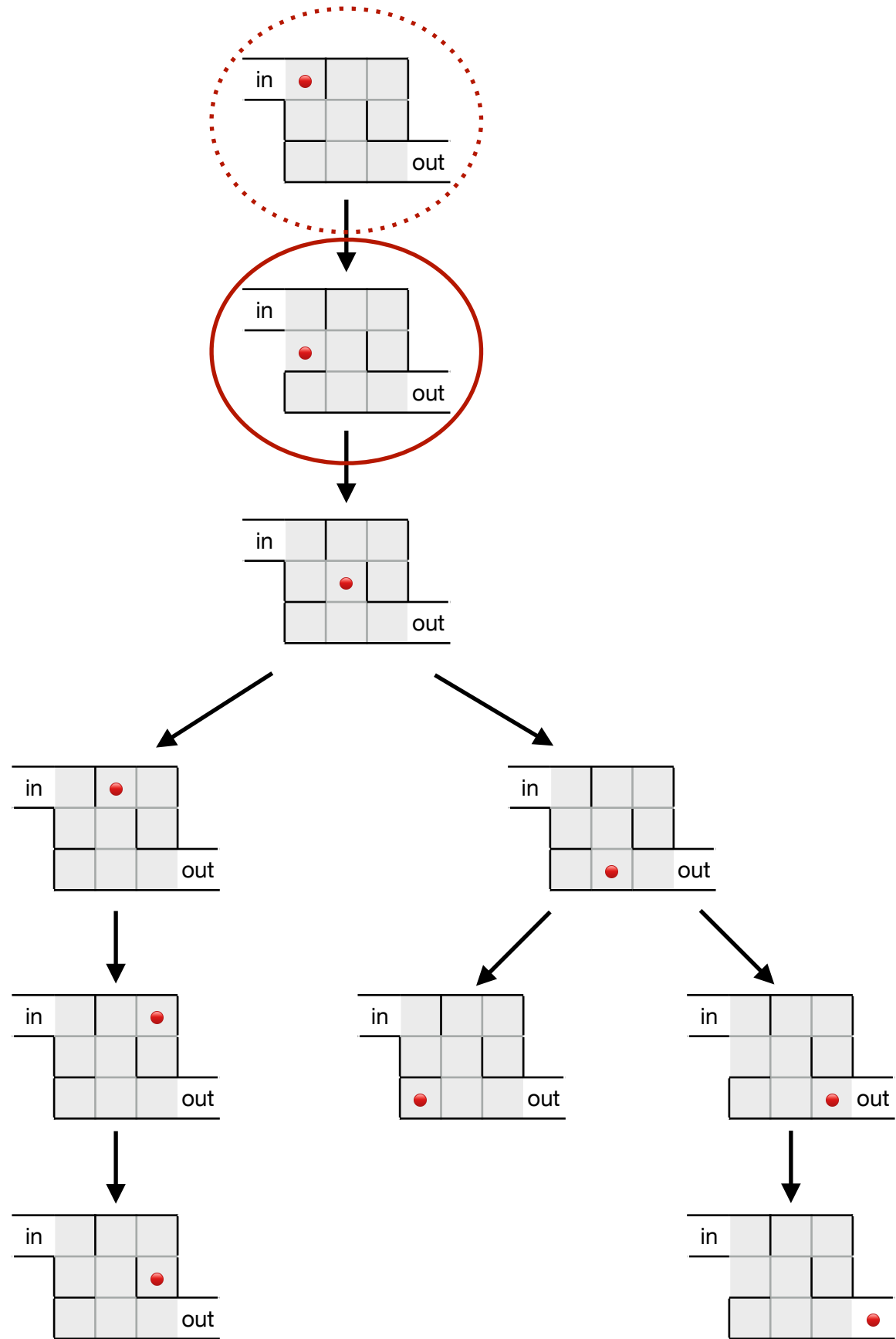
# Maze “space tree”

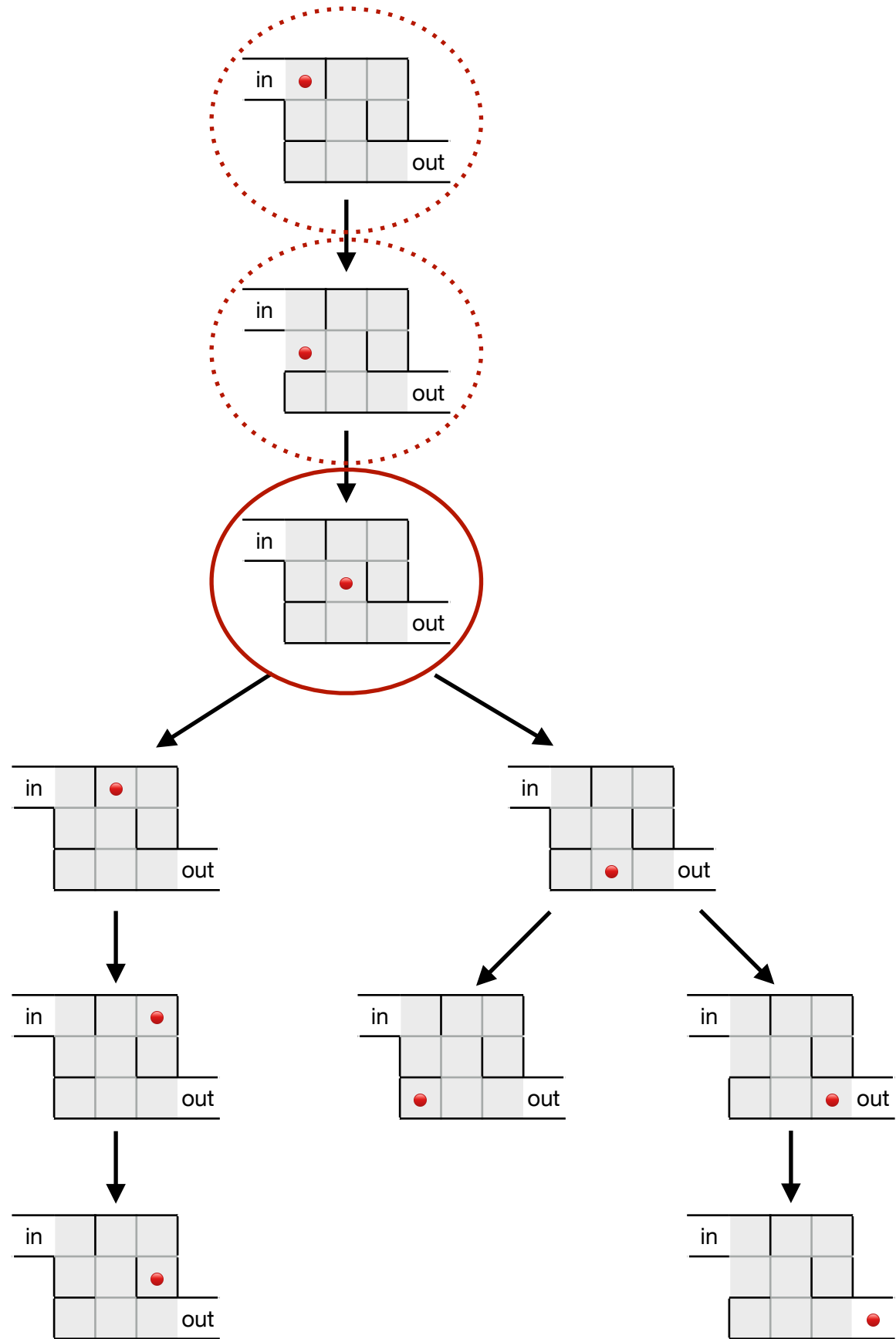


# Depth-first search

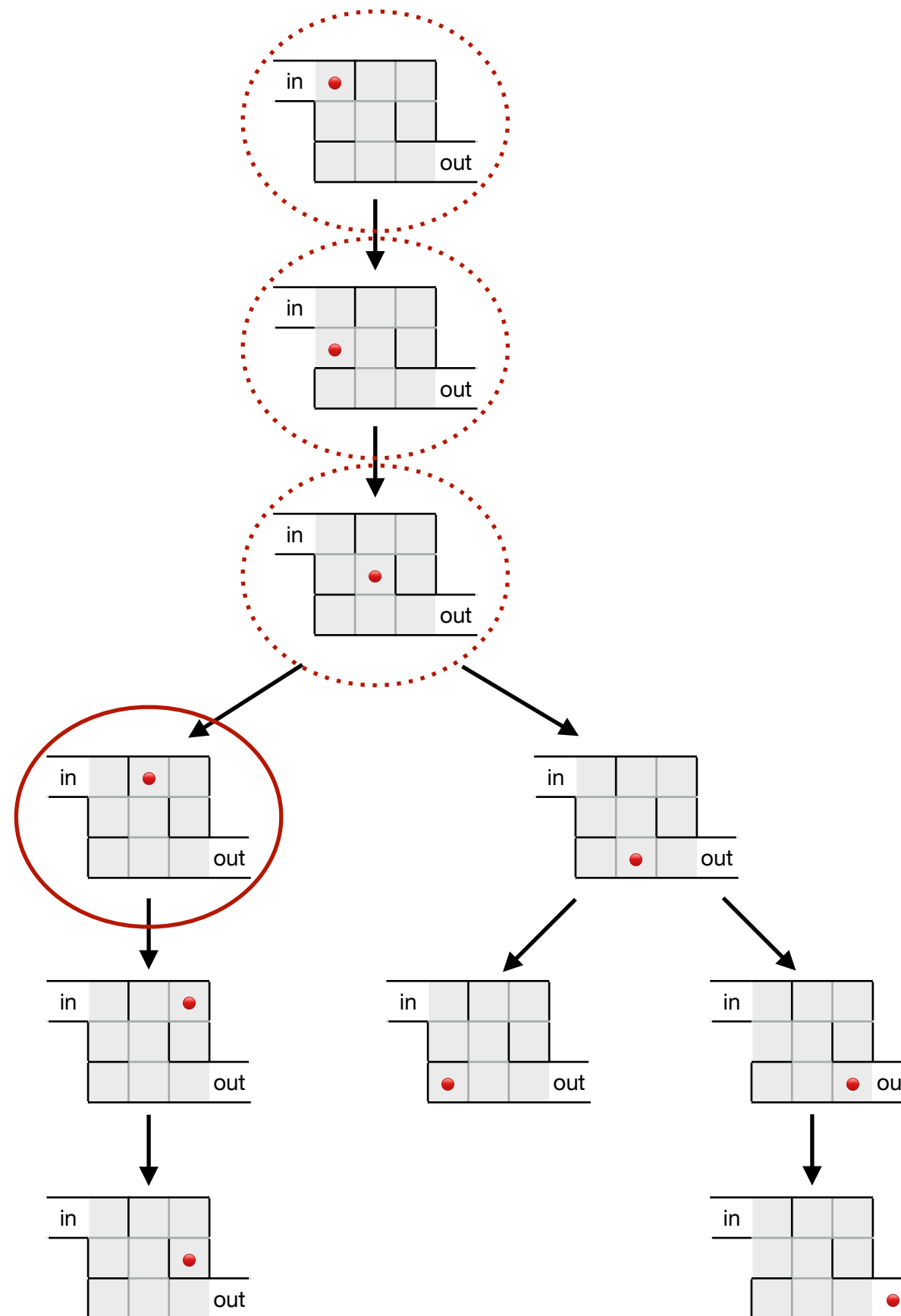


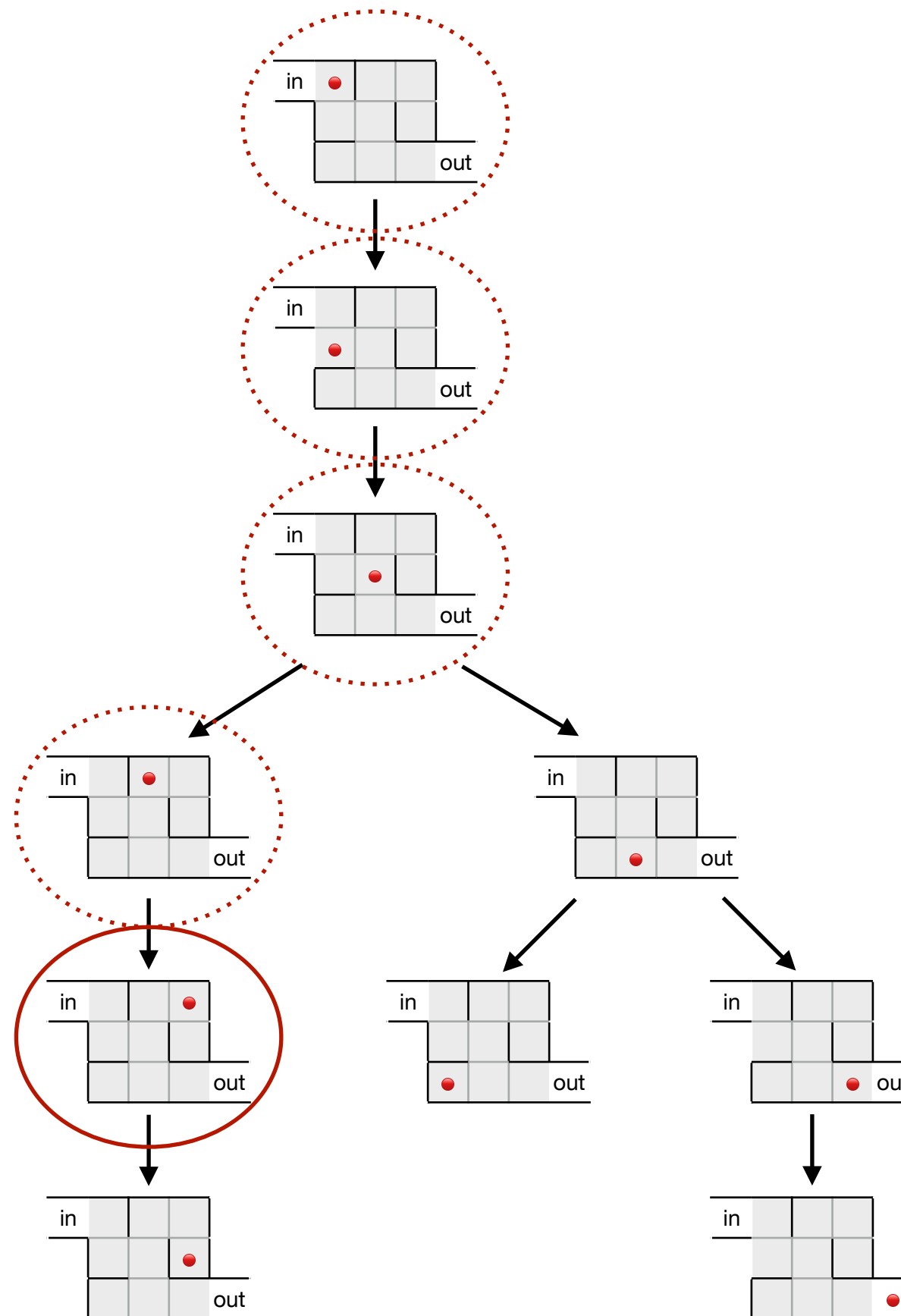






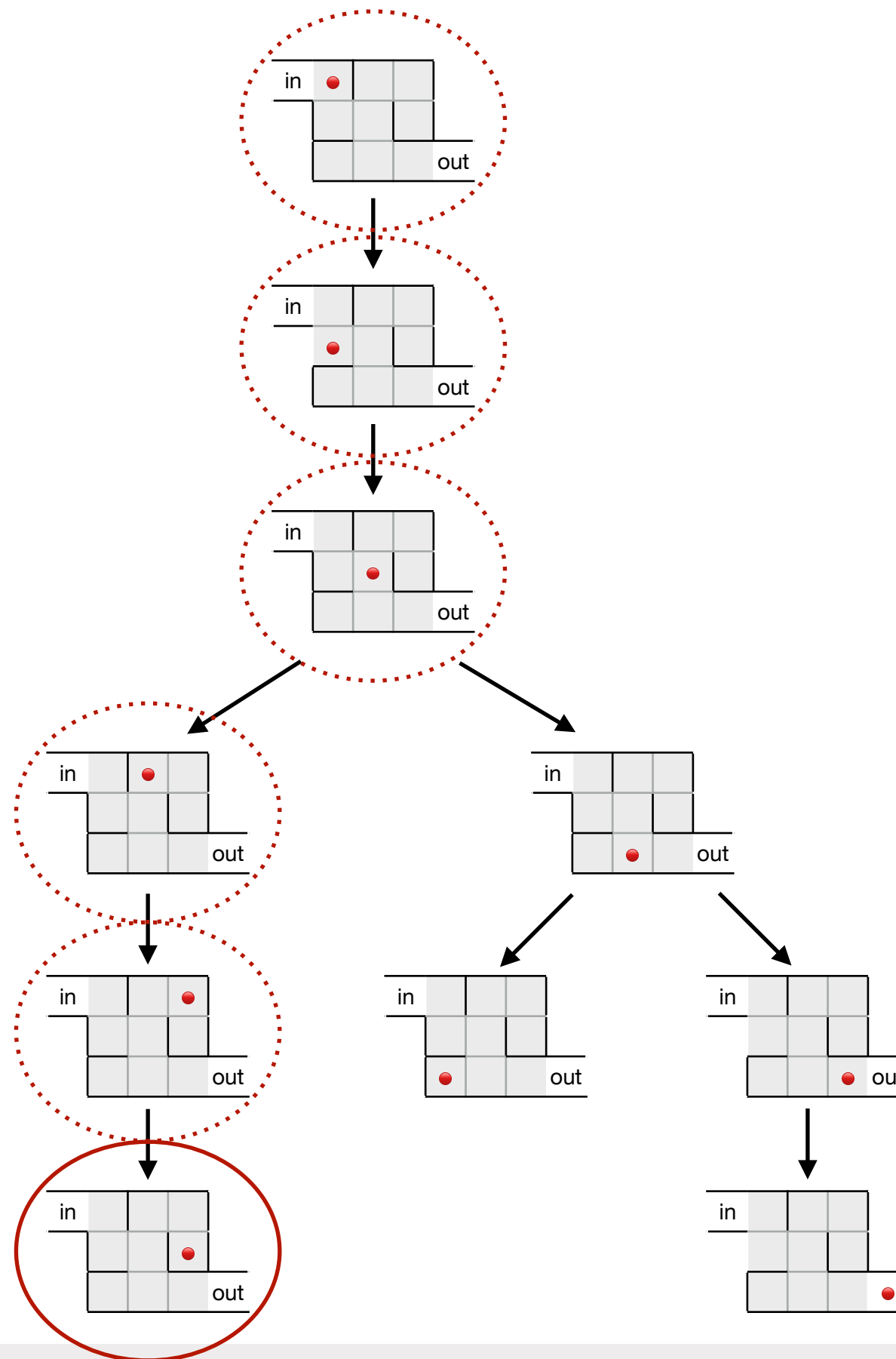




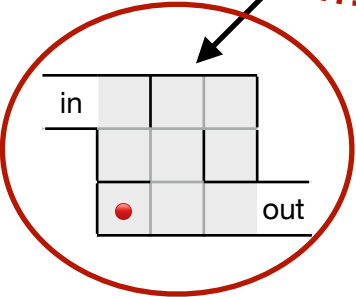


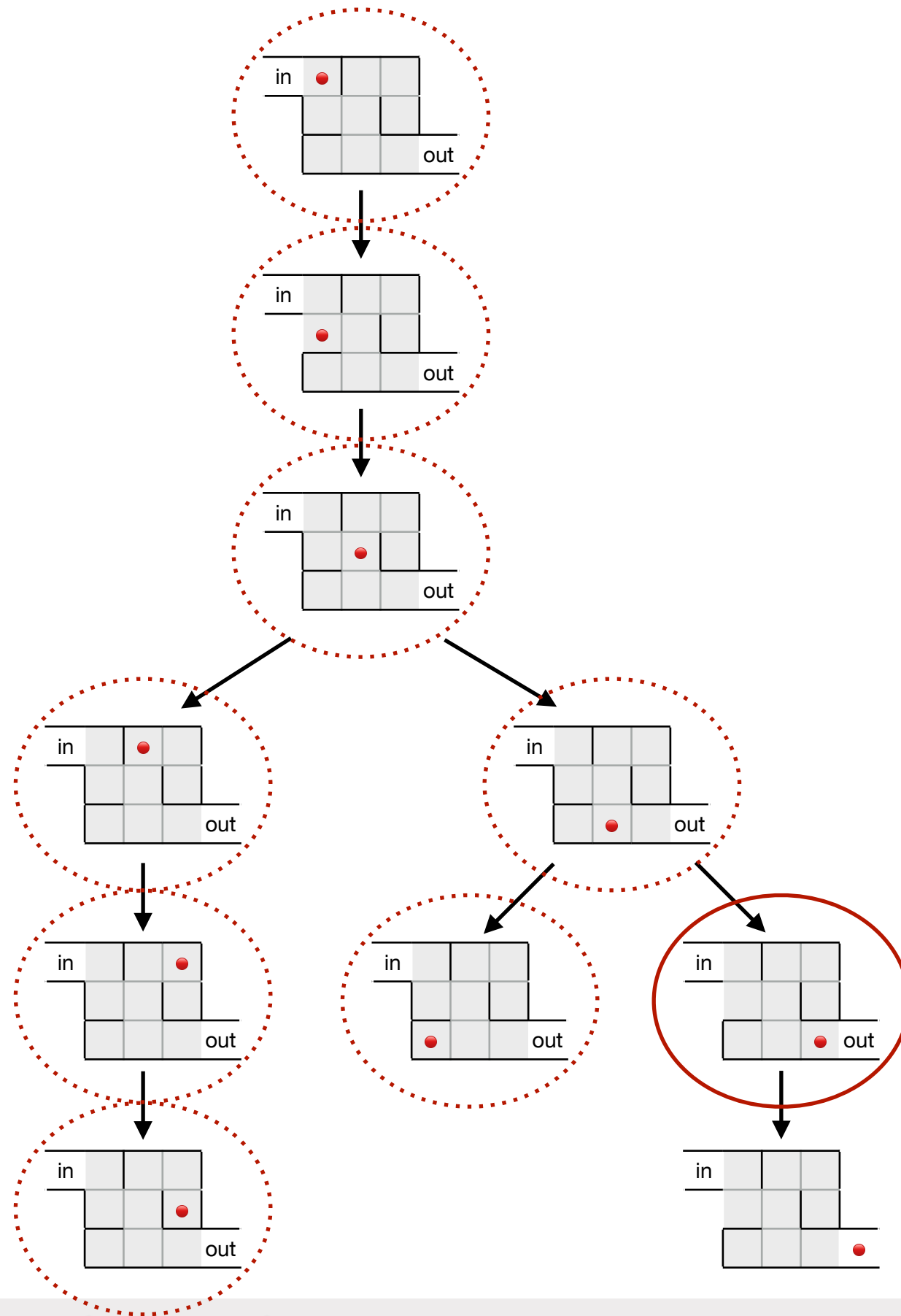
AIgaming.com

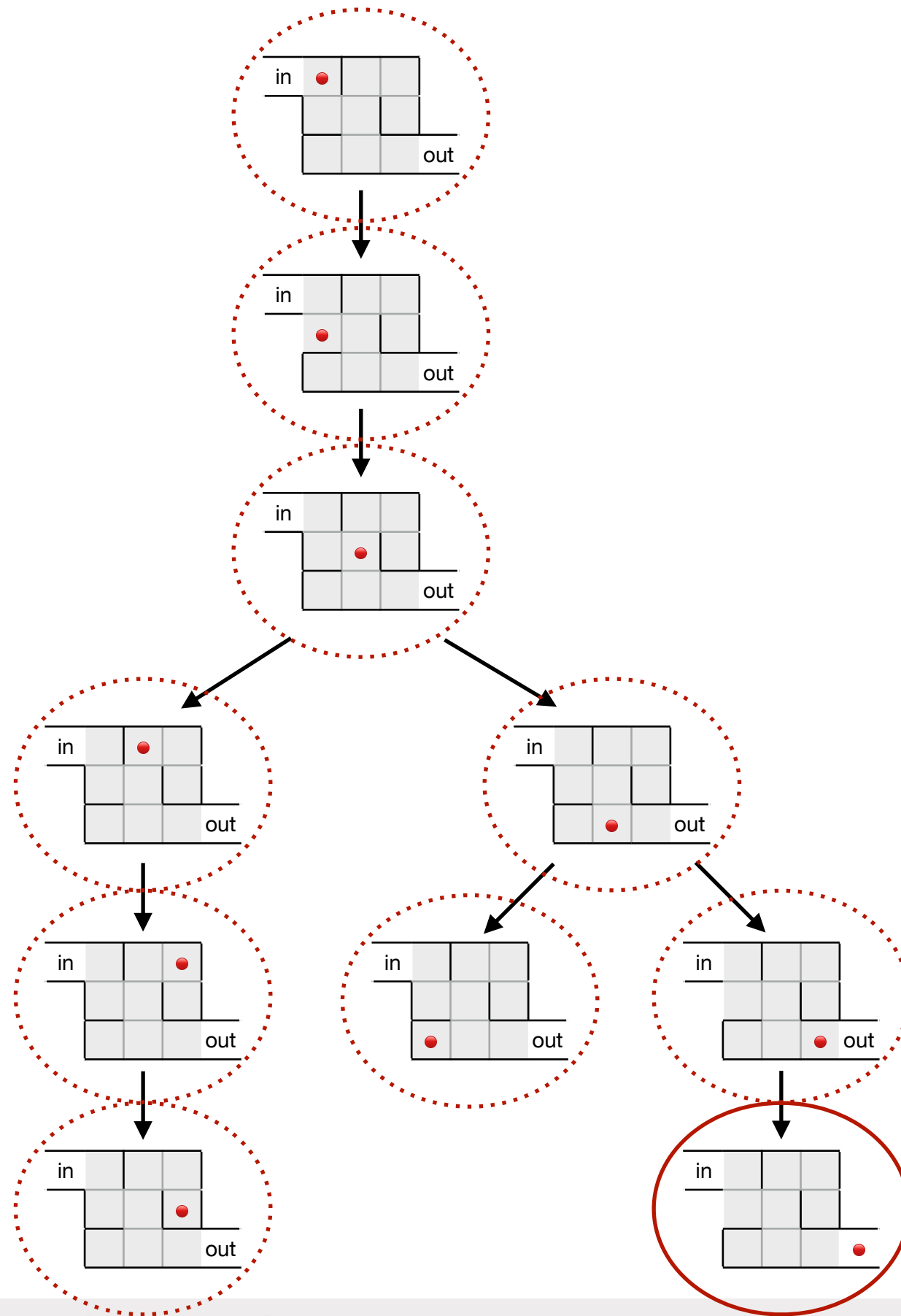












# DFS algorithm

**DFS:** given a graph **G** and a starting node **n**

0. put **n** into a *stack*
1. *pop top node* from the stack and mark it as *visited*
2. put all nodes reachable from the top node in the *stack*
3. while the stack is not empty, recursively apply steps 1–3



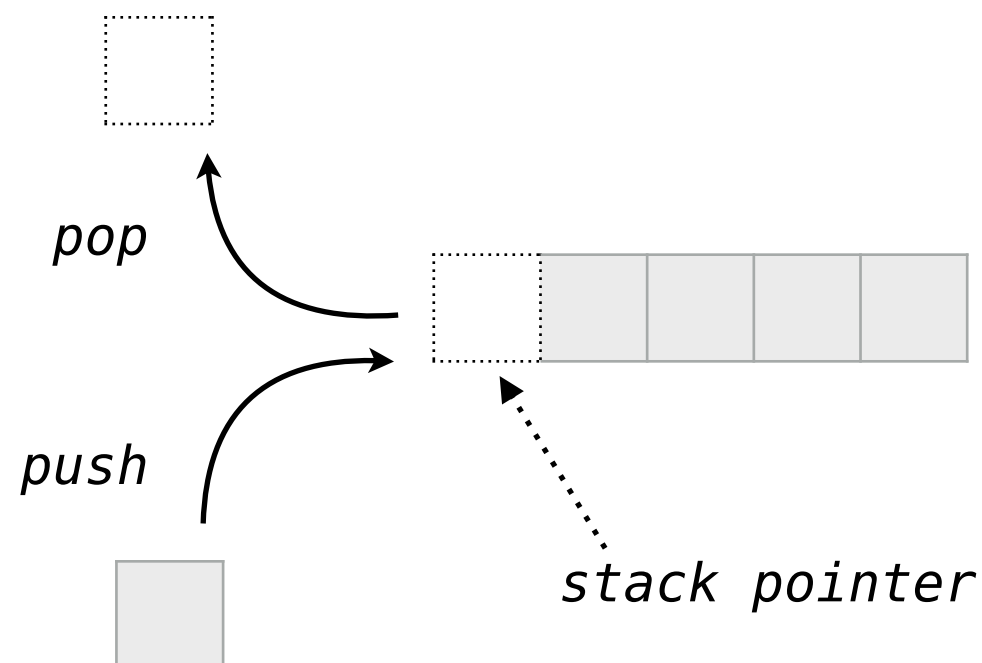


# DFS algorithm

**DFS:** given a graph **G** and a starting node **n**

0. put **n** into a *stack*
1. *pop top node* from the stack and mark it as *visited*
2. put all nodes reachable from the top node in the *stack*
3. while the stack is not empty, recursively apply steps 1–3

**Stack:**



**LIFO:**  
last in,  
first out



# DFS algorithm

```
def dfs(graph, start):  
    visited, stack = set(), [start]  
    while stack:  
        vertex = stack.pop()  
        if vertex not in visited:  
            visited.add(vertex)  
  
            print(vertex)
```



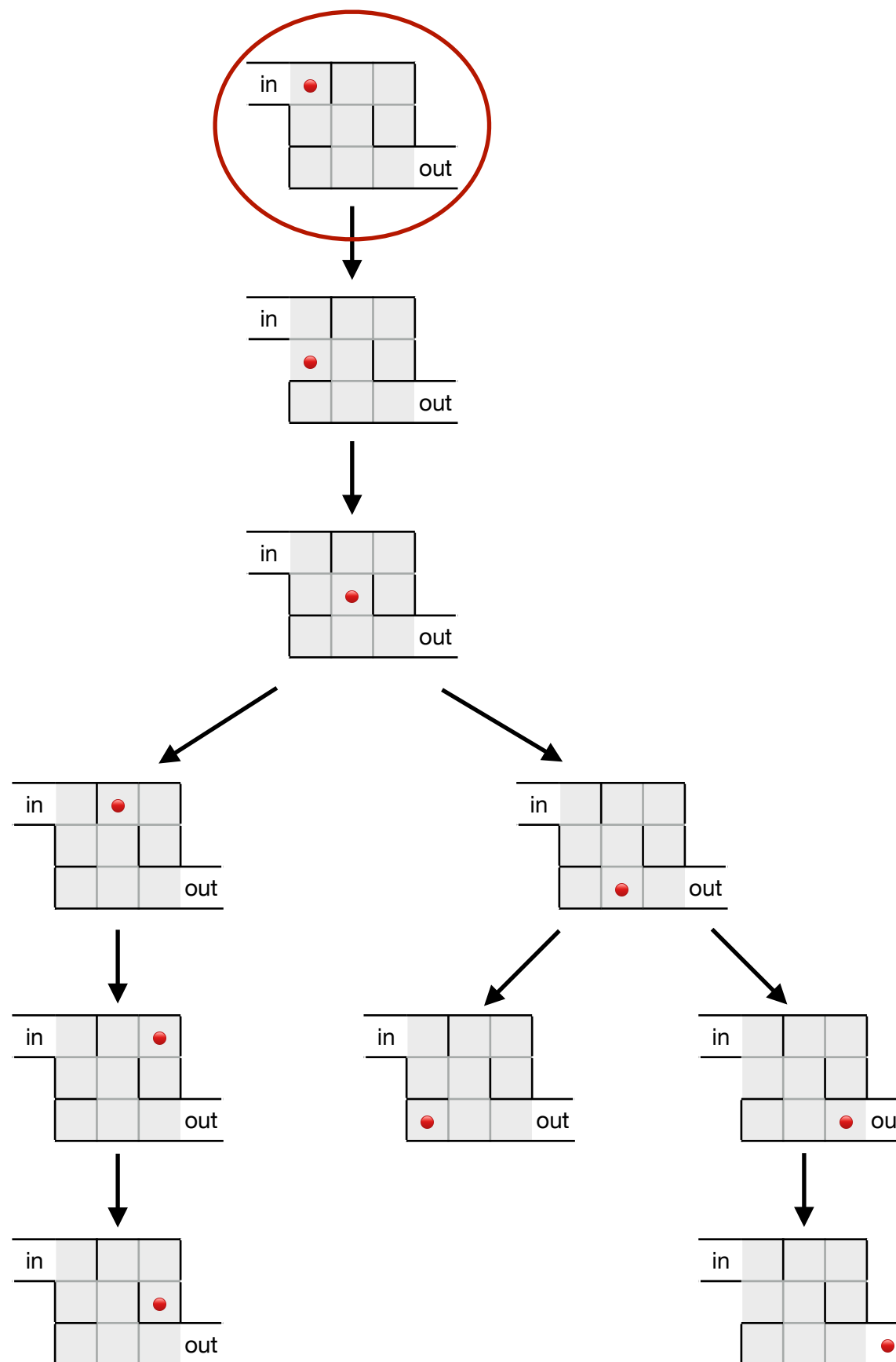
# DFS algorithm

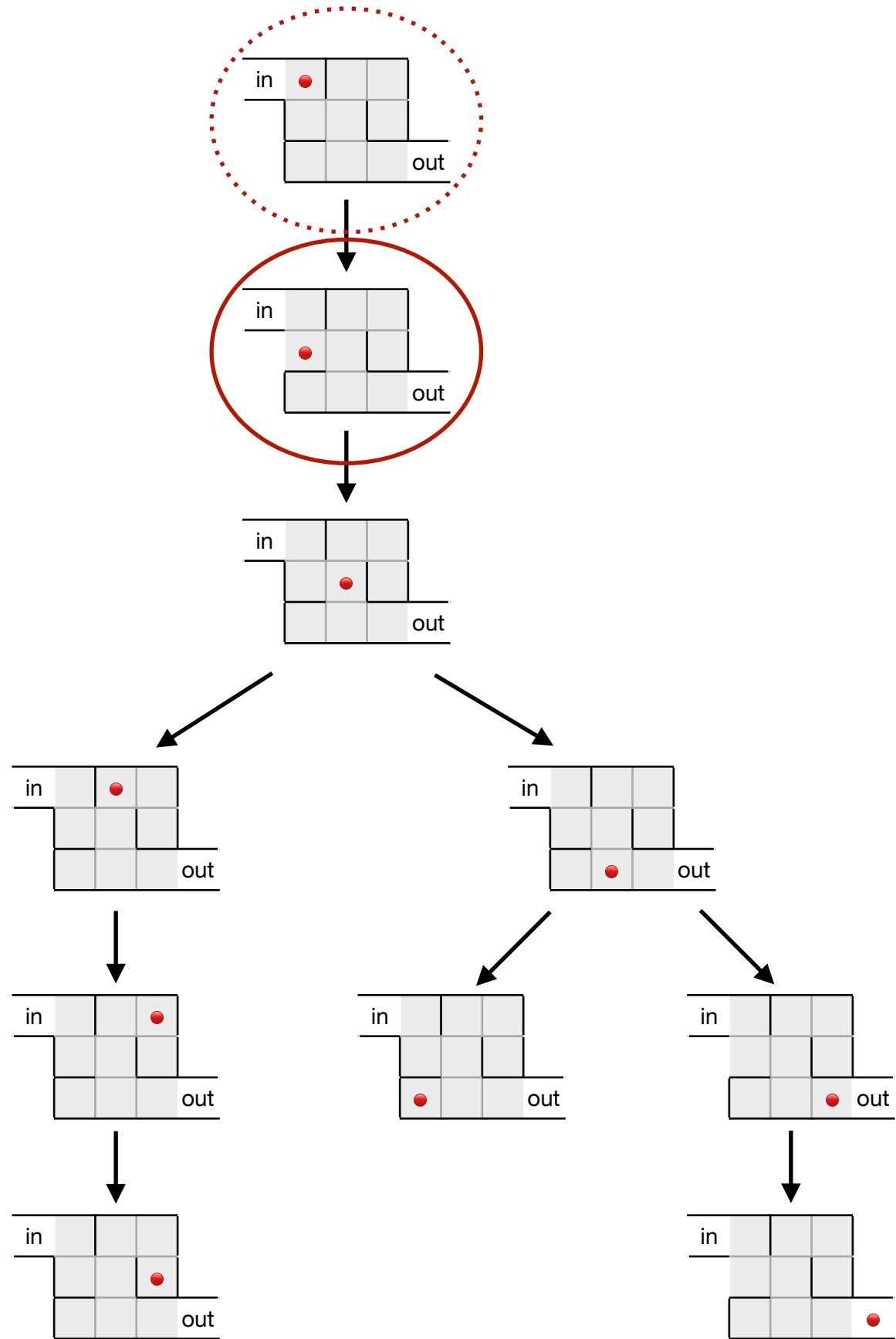
```
def dfs(graph, start):  
    visited, stack = set(), [start]  
    while stack:  
        vertex = stack.pop()  
        if vertex not in visited:  
            visited.add(vertex)  
            stack.extend(graph[vertex] - visited)  
            print(vertex)
```

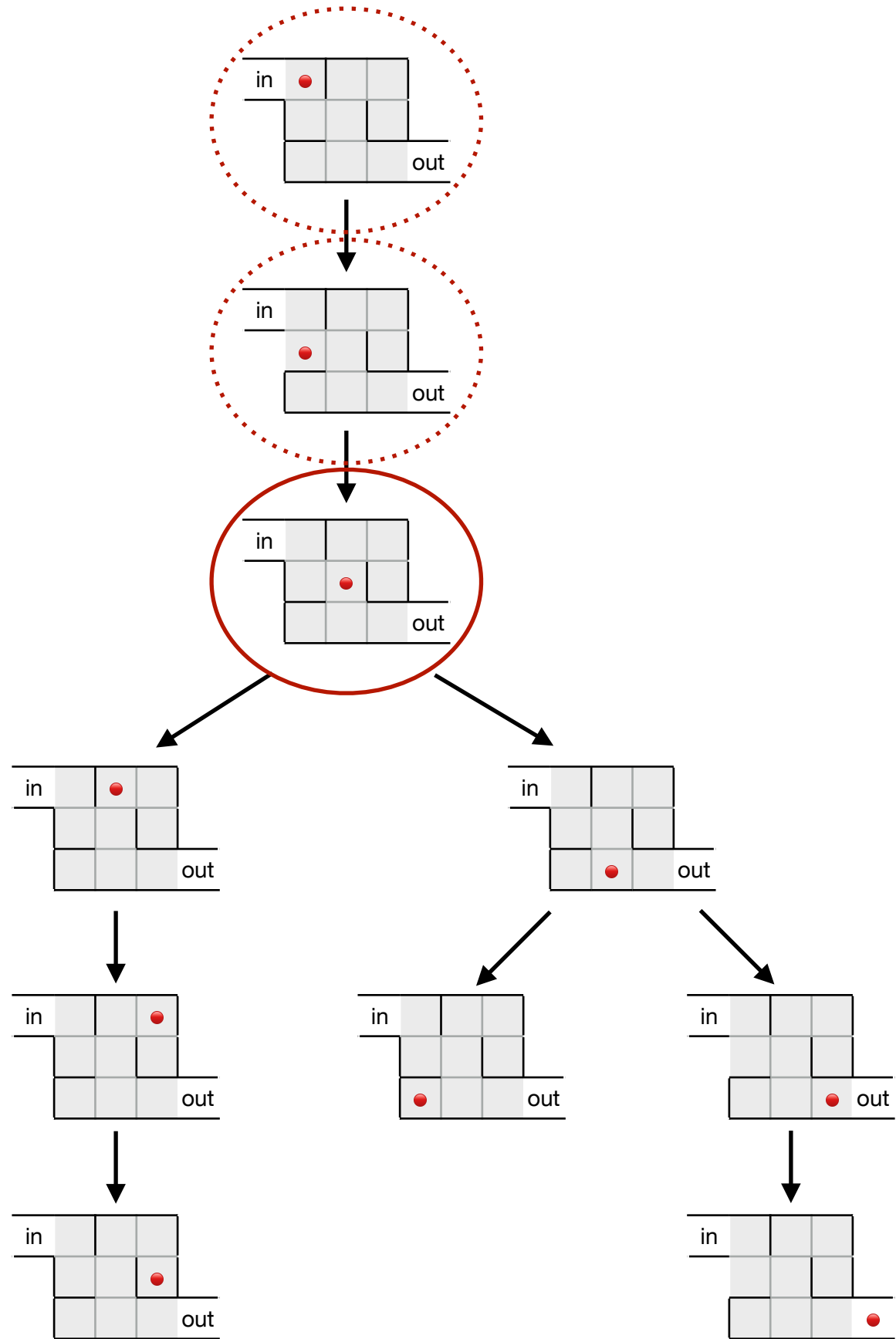


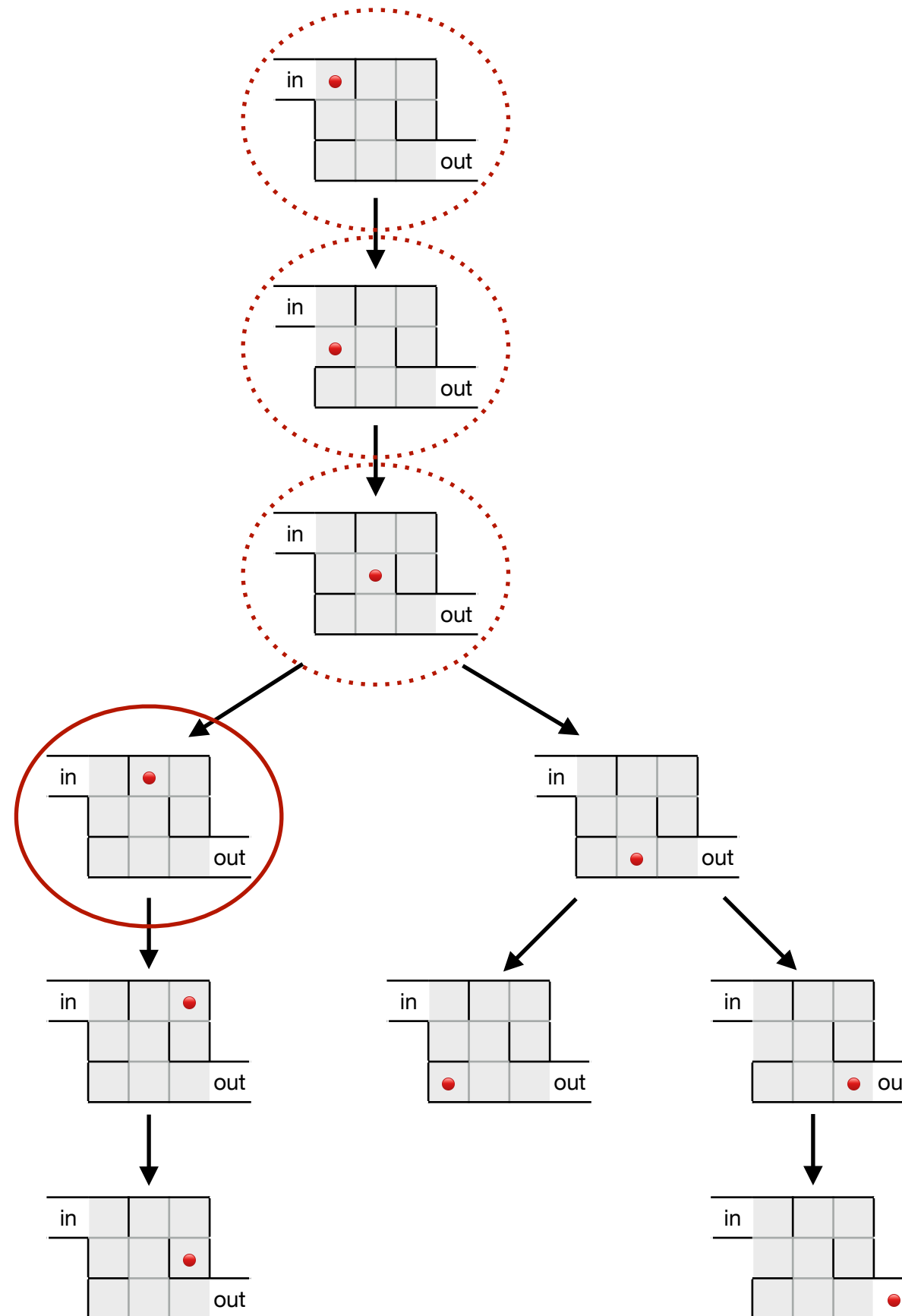
# Breadth-first search



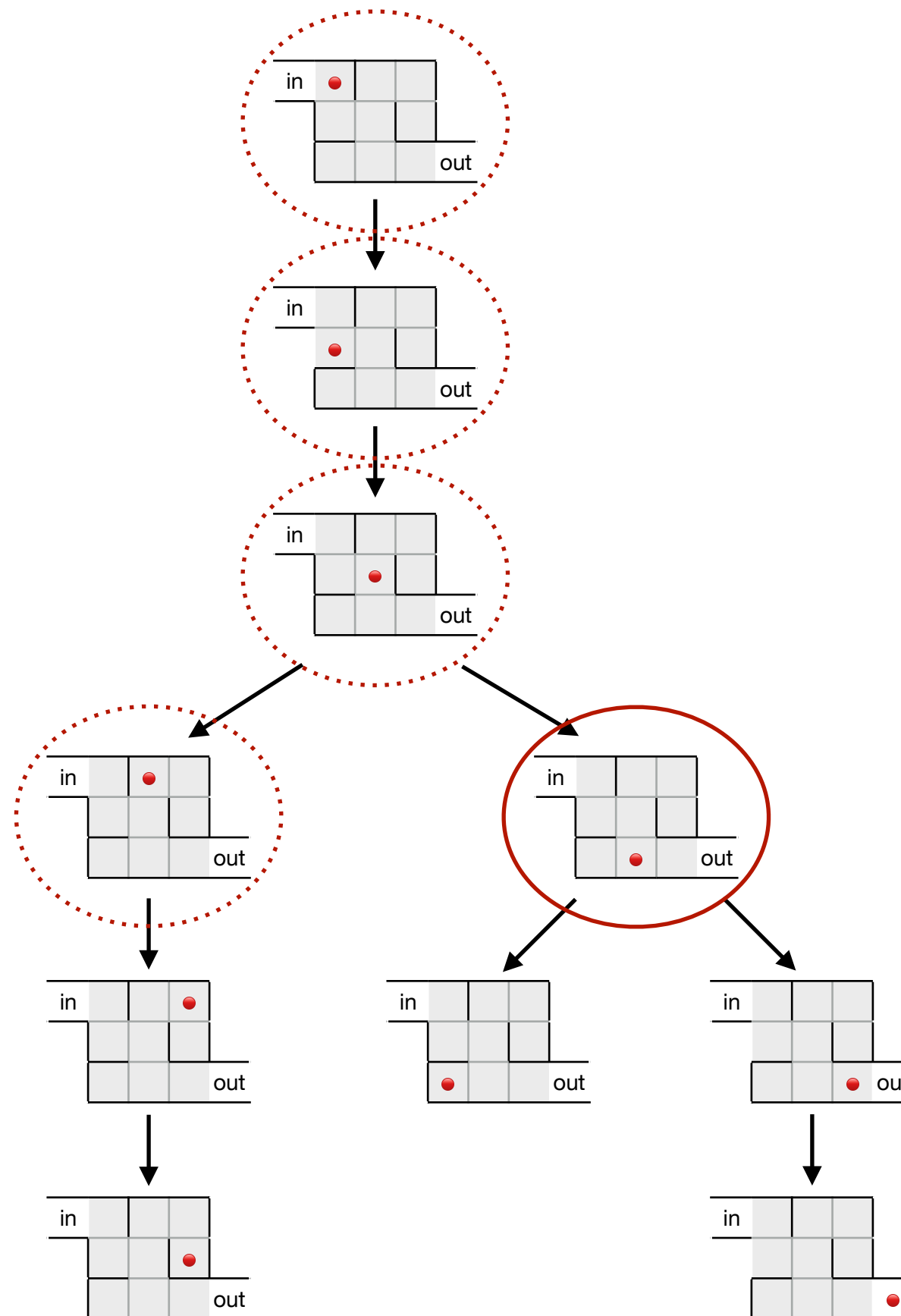


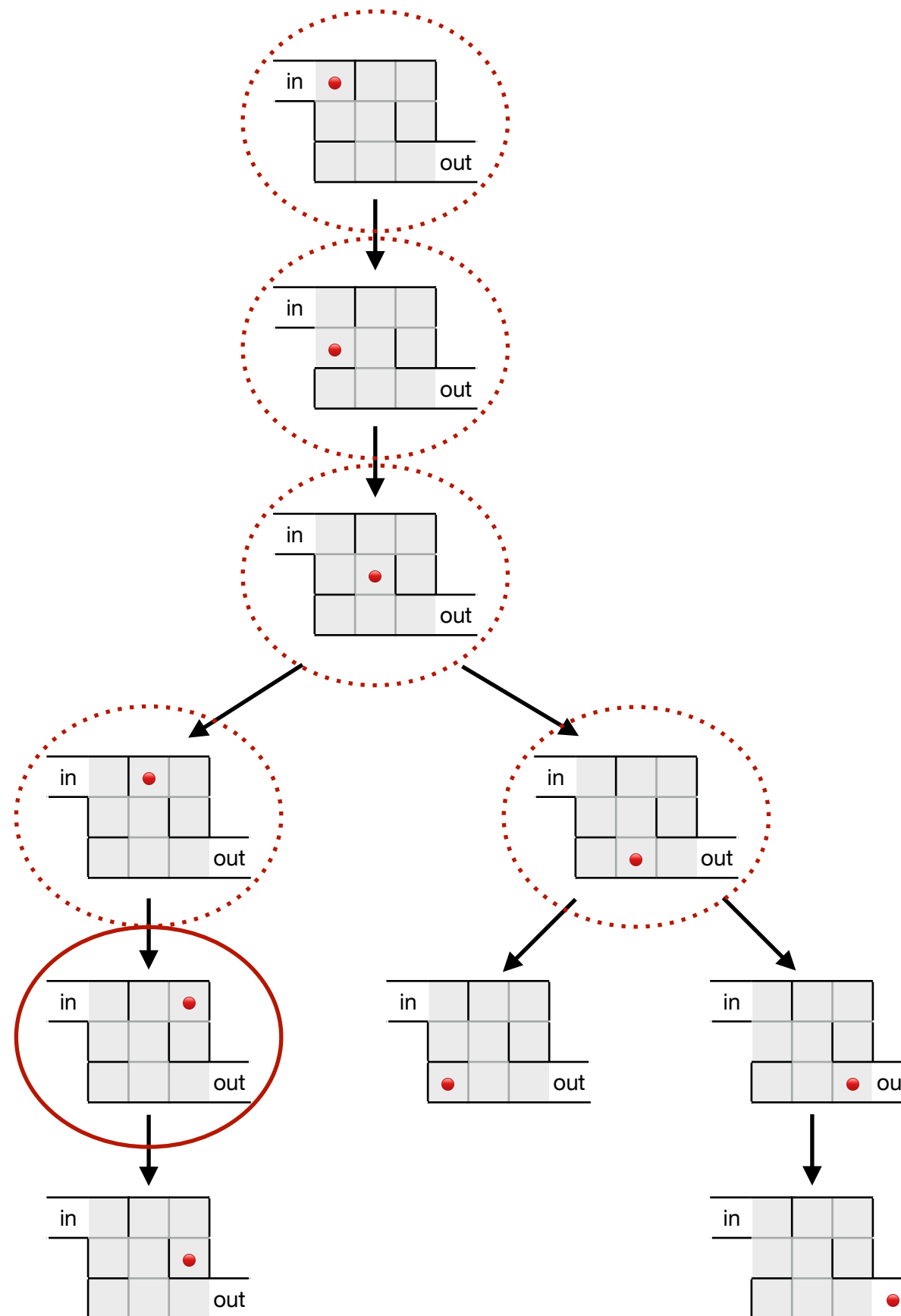


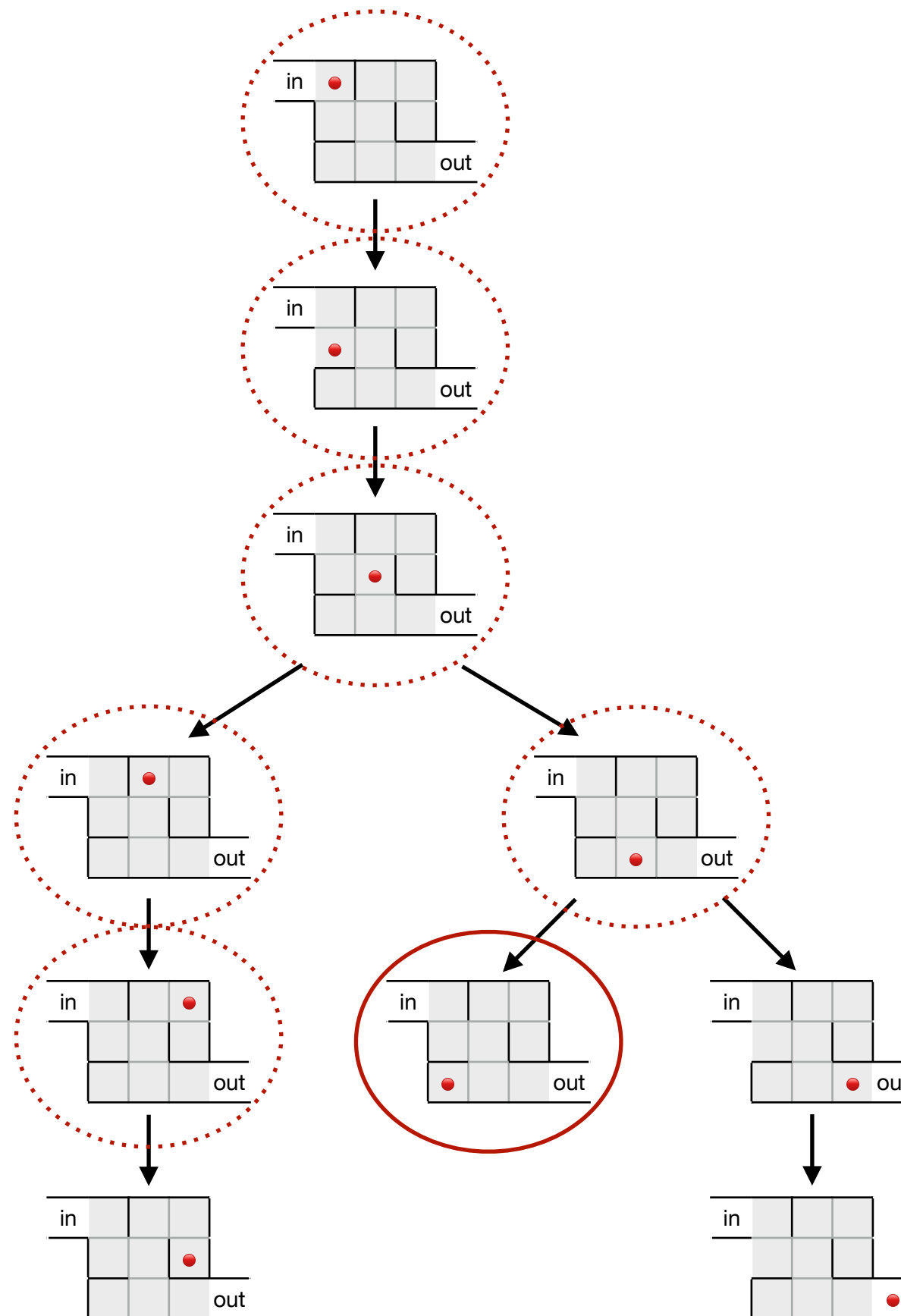


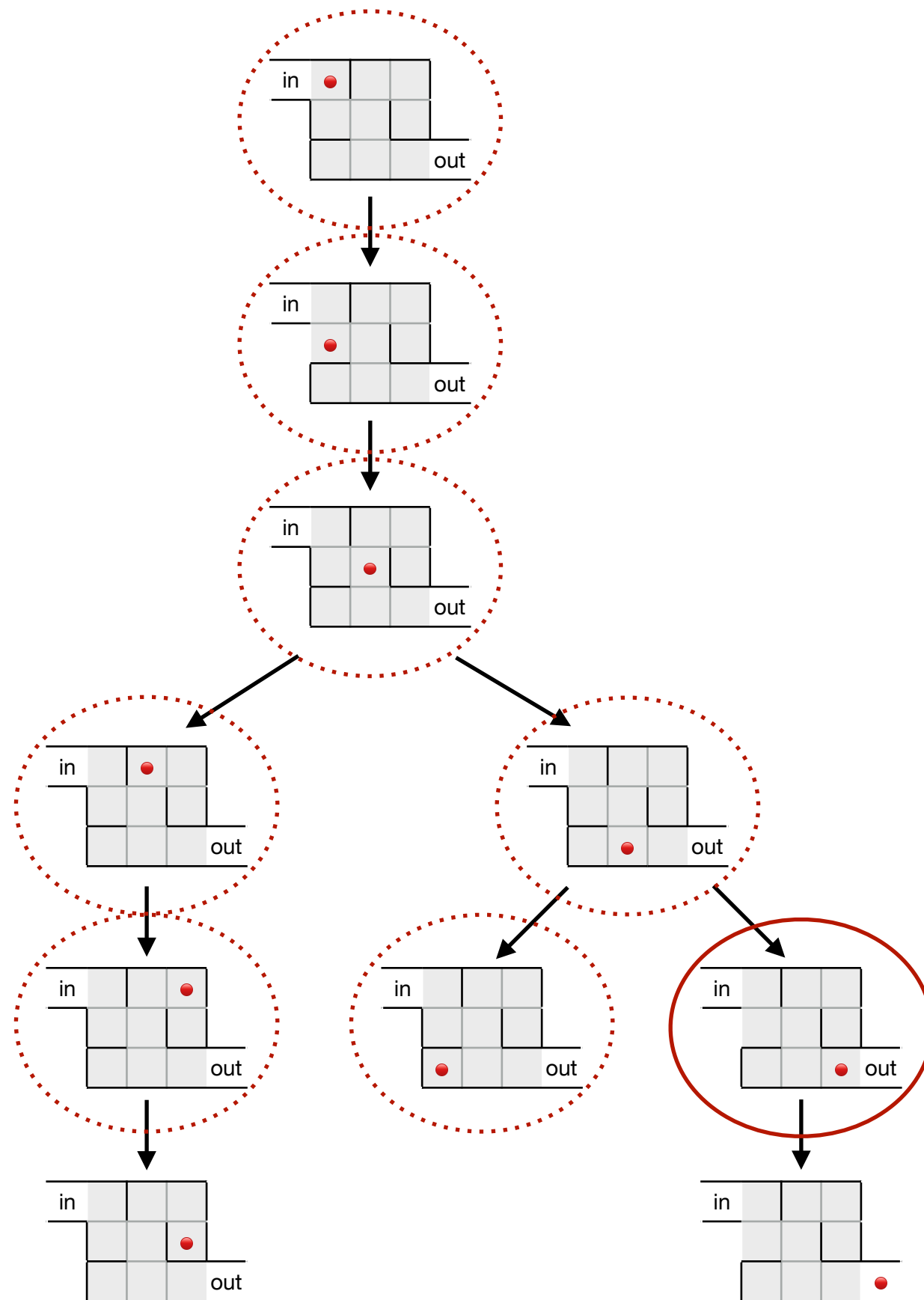


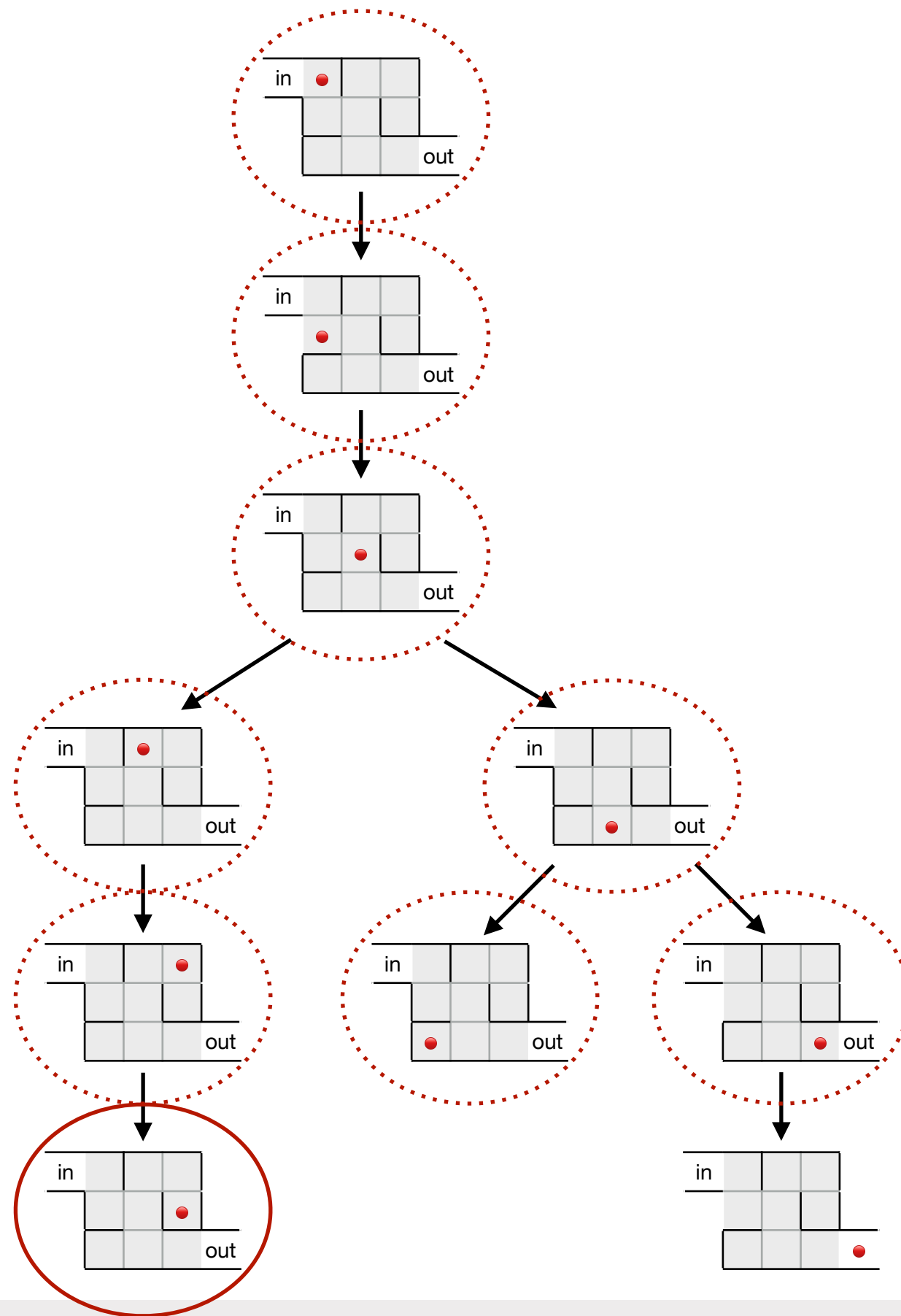














# BFS algorithm

**BFS:** given a graph **G** and a starting node **n**

0. put **n** into a *queue*
1. *dequeue first node* from the queue and mark it as *visited*
2. put all nodes reachable from the first node in the *queue*
3. while the queue is not empty, recursively apply steps 1–3

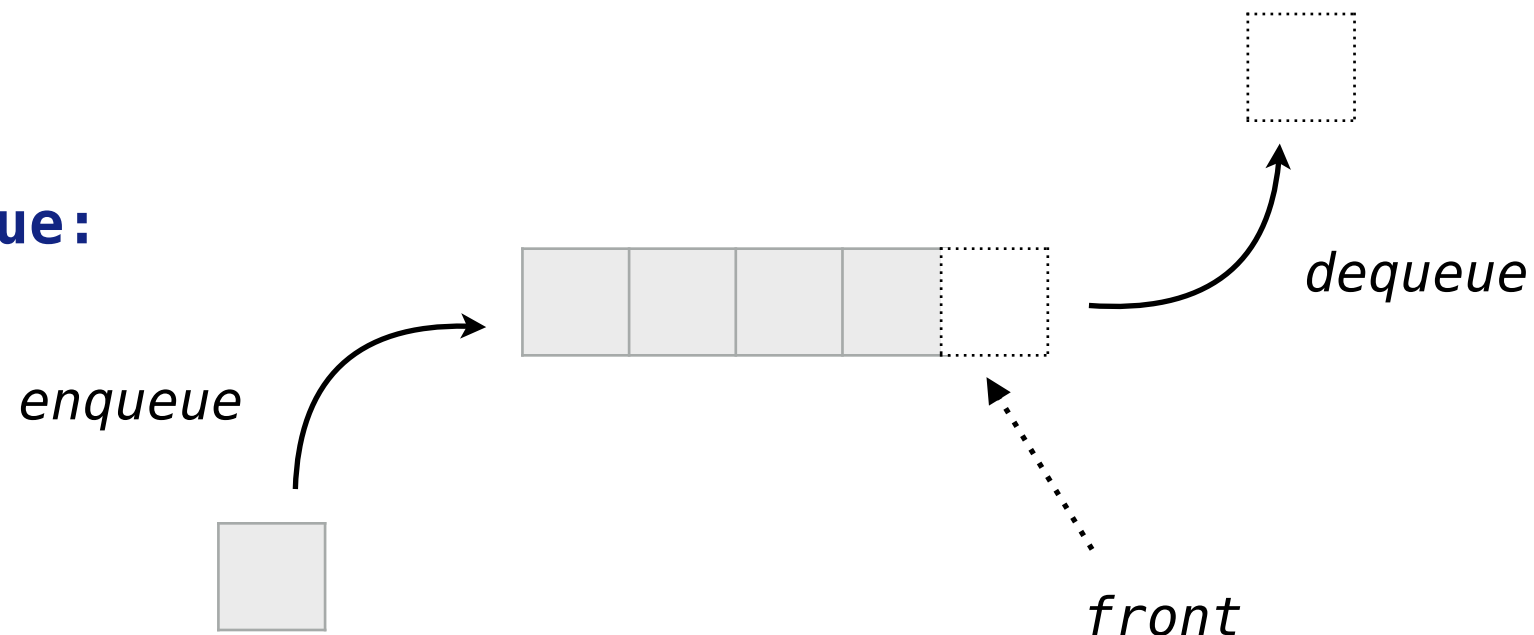


# BFS algorithm

**BFS:** given a graph **G** and a starting node **n**

0. put **n** into a *queue*
1. *dequeue first node* from the queue and mark it as *visited*
2. put all nodes reachable from the first node in the *queue*
3. while the queue is not empty, recursively apply steps 1–3

**Queue:**



**FIFO:**  
first in,  
first out





# BFS algorithm

```
def bfs(graph, start):  
    visited, queue = set(), [start]  
    while queue:  
        vertex = queue.pop(0)  
        if vertex not in visited:  
            visited.add(vertex)  
  
            print(vertex)
```



# BFS algorithm

```
def bfs(graph, start):  
    visited, queue = set(), [start]  
    while queue:  
        vertex = queue.pop(0)  
        if vertex not in visited:  
            visited.add(vertex)  
            queue.extend(sorted(graph[vertex] - visited))  
            print(vertex)
```



# Search in Battleship



AIgaming.com



# Target vs Hunt mode

- in the *Hunt* mode, we randomly search for ships on the board:

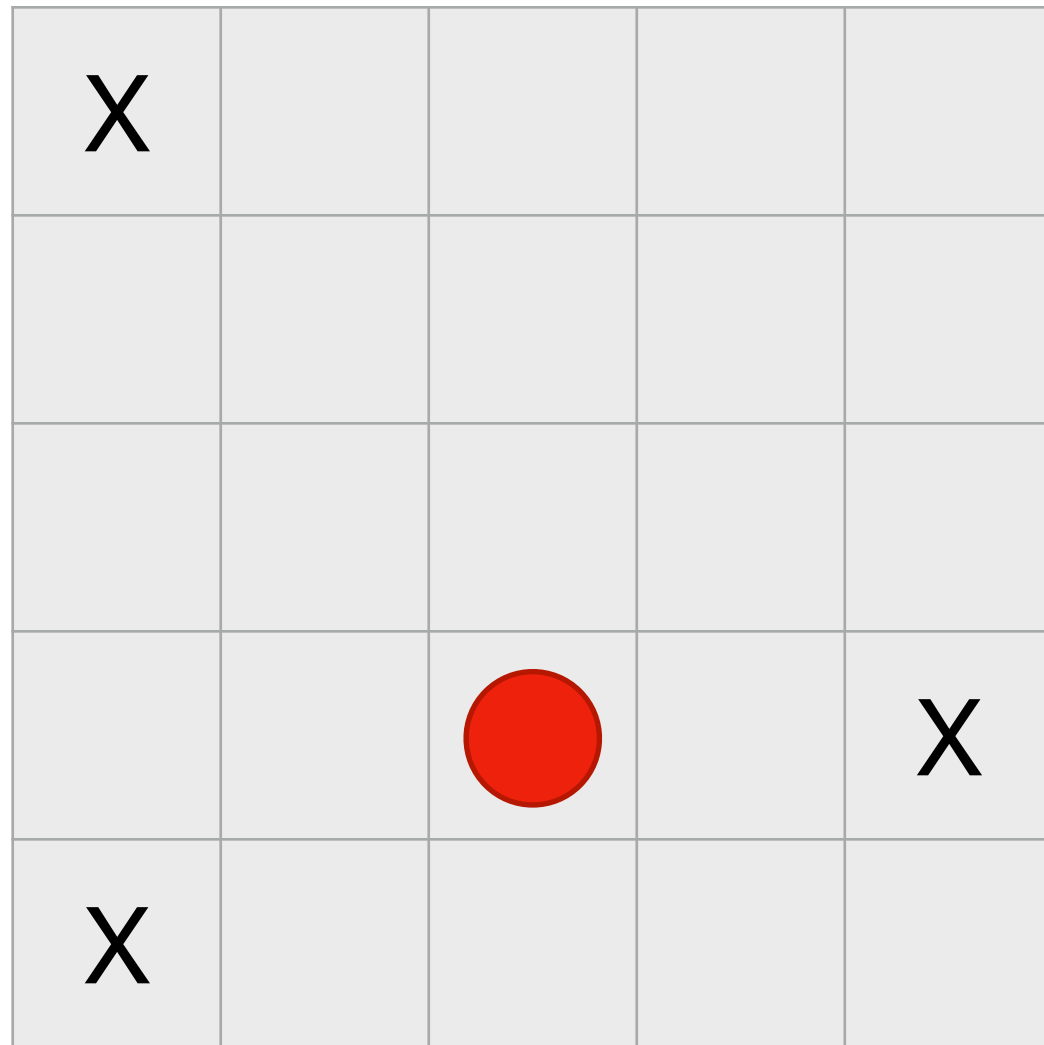
*chooseRandomValidTarget(gamestate)*

- in the *Target* mode, when a ship is hit, we try to sink it by searching through its neighbourhood cells!

*which type of search should we choose?*



# Search in Target mode



X is a missed shot,

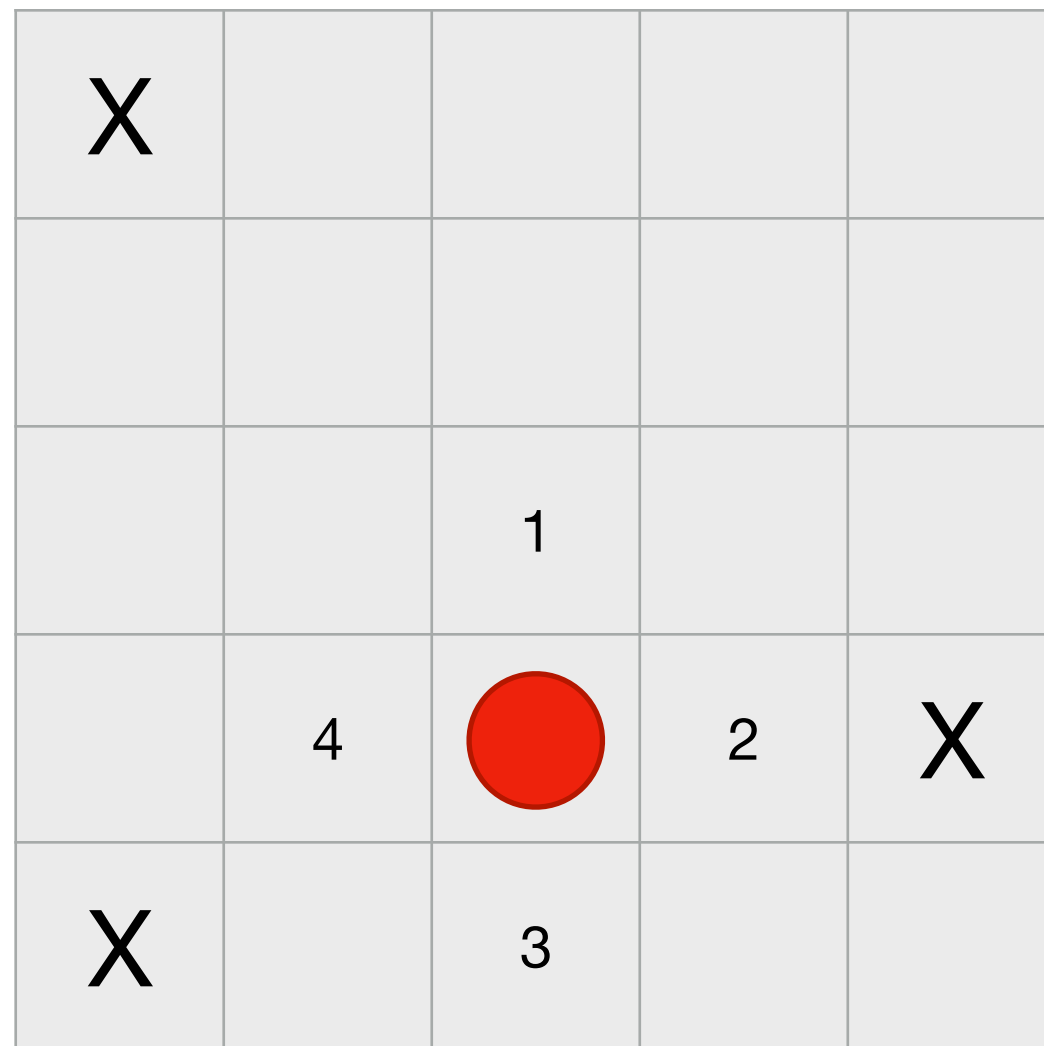
● is a hit ship.



AIgaming.com



# Search in Target mode



**BFS** will first consider all neighbours of the start node, even if the next hit is discovered.



AIgaming.com



# Search in Target mode

X				
		●		
	4	●	2	X
X		3		


**BFS** will first consider all neighbours of the start node, even if the next hit is discovered.



AIgaming.com



# Search in Target mode


X		3	4	5
		2		
		1		
				X
X				

**DFS**, on the other hand, will follow consider all neighbours of the start node, even if the next hit is discovered.






# Search in Target mode

X		3	4	5
		2		
		X		
				X
X				

**DFS**, on the other hand, will follow consider all neighbours of the start node, even if the next hit is discovered.



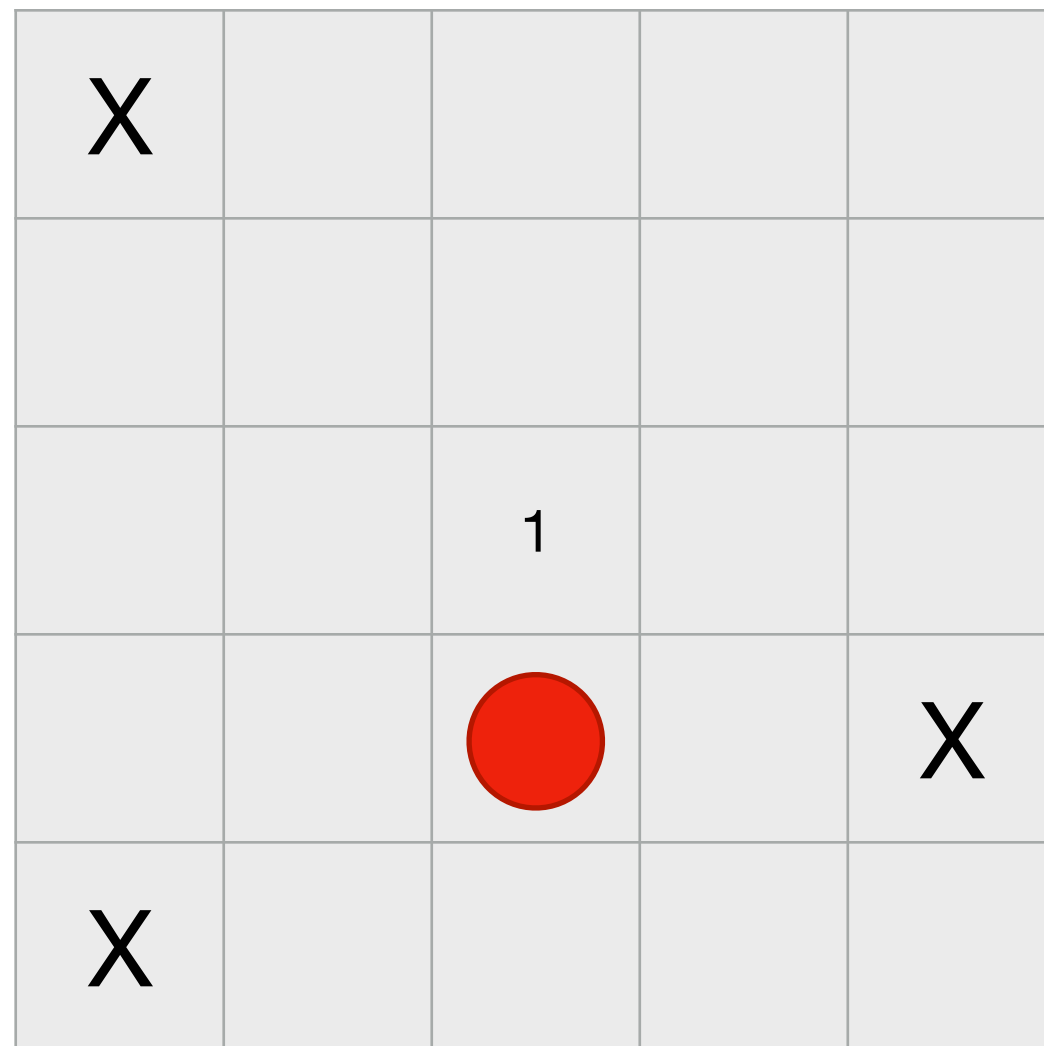
# Search in Target mode

X		3	4	5
		X		
		X		
				X
X				

**DFS**, on the other hand, will follow consider all neighbours of the start node, even if the next hit is discovered.



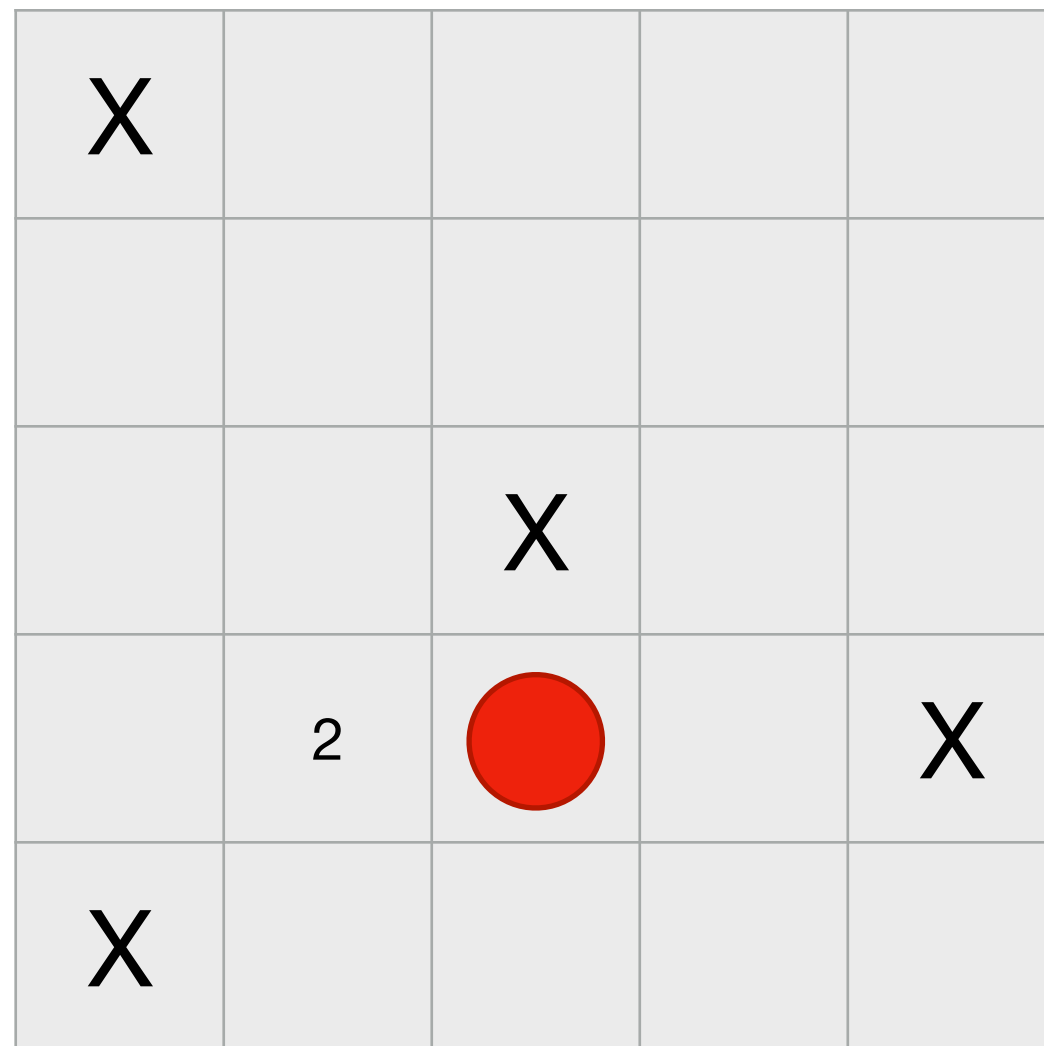
# Search in Target mode



**Solution: DFS with pruning**  
whenever the top node returns a miss, we do not add its neighbours to the stack.



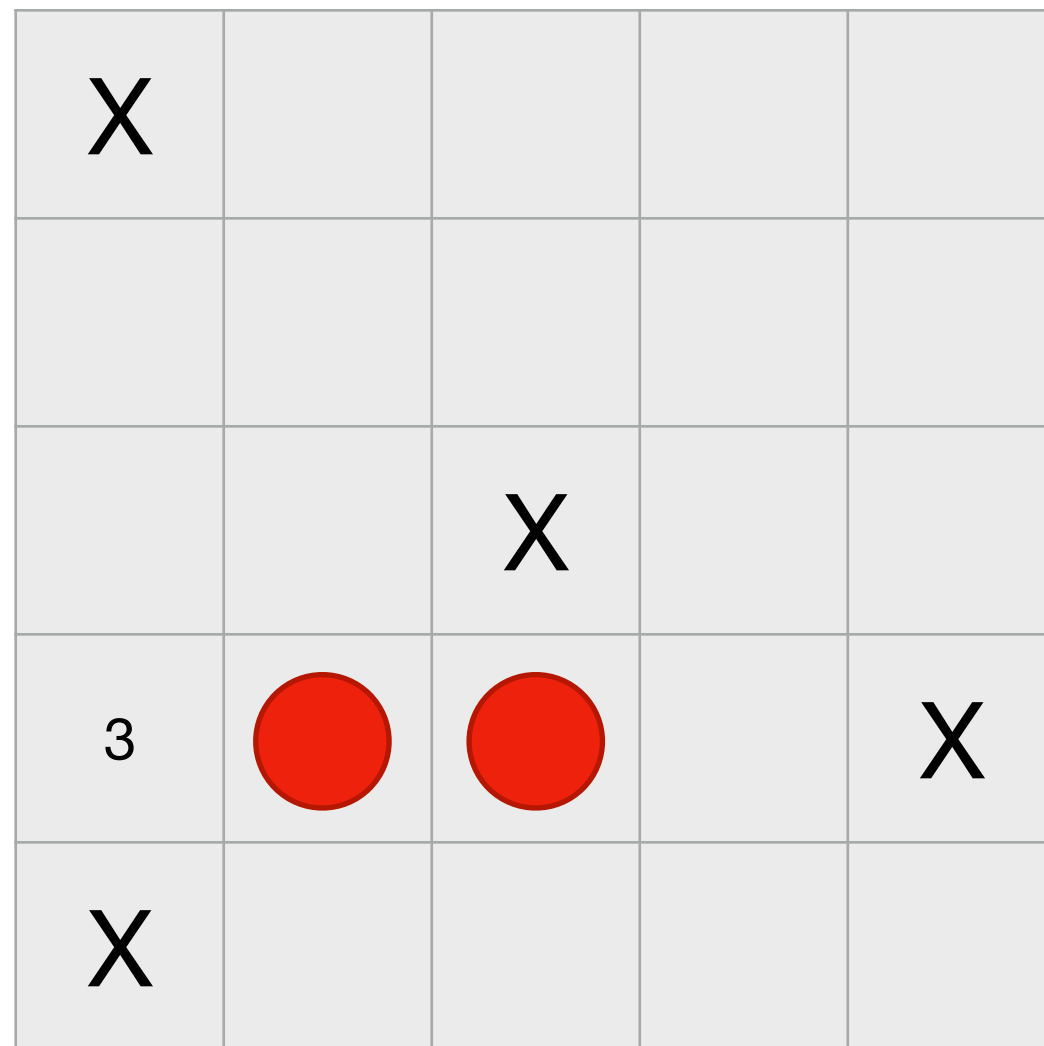
# Search in Target mode



**Solution: DFS with pruning**  
whenever the top node returns a miss, we do not add its neighbours to the stack.



# Search in Target mode



**Solution: DFS with pruning**  
whenever the top node returns a miss, we do not add its neighbours to the stack.



# Coding search strategies



46



AIgaming.com



# Using the online editor

- **Problem:** in the online editor you cannot *memorize* the previous search state, in particular, your previous moves.
- Hence, you do not know where to *backtrack*.

## You only get the 'GameState':

```
{'Ships': [5, 4, 3, 3, 2], 'IsMover': True, 'Round': 5, 'GameStatus': 'RUNNING', 'GameId': 123, 'OpponentId': 'enemy',  
 'MyBoard': [[[' ', '2', '2', '2', ' ', ' ', ' ', ' ', ' ', ' '], [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '], ['0', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
               ['0', '4', '4', ' ', ' ', ' ', ' ', ' ', ' ', ' '], ['0', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '3'], ['0', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '3'], ['0', ' ', ' ',  
               ' ', ' ', ' ', '3'], [' ', '1', '1', '1', '1', ' ', ' ', ' ', ' ']],  
 'OppBoard': [[[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '], [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '], [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
                [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '], [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '], [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
                [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']]}
```

# Using the online editor

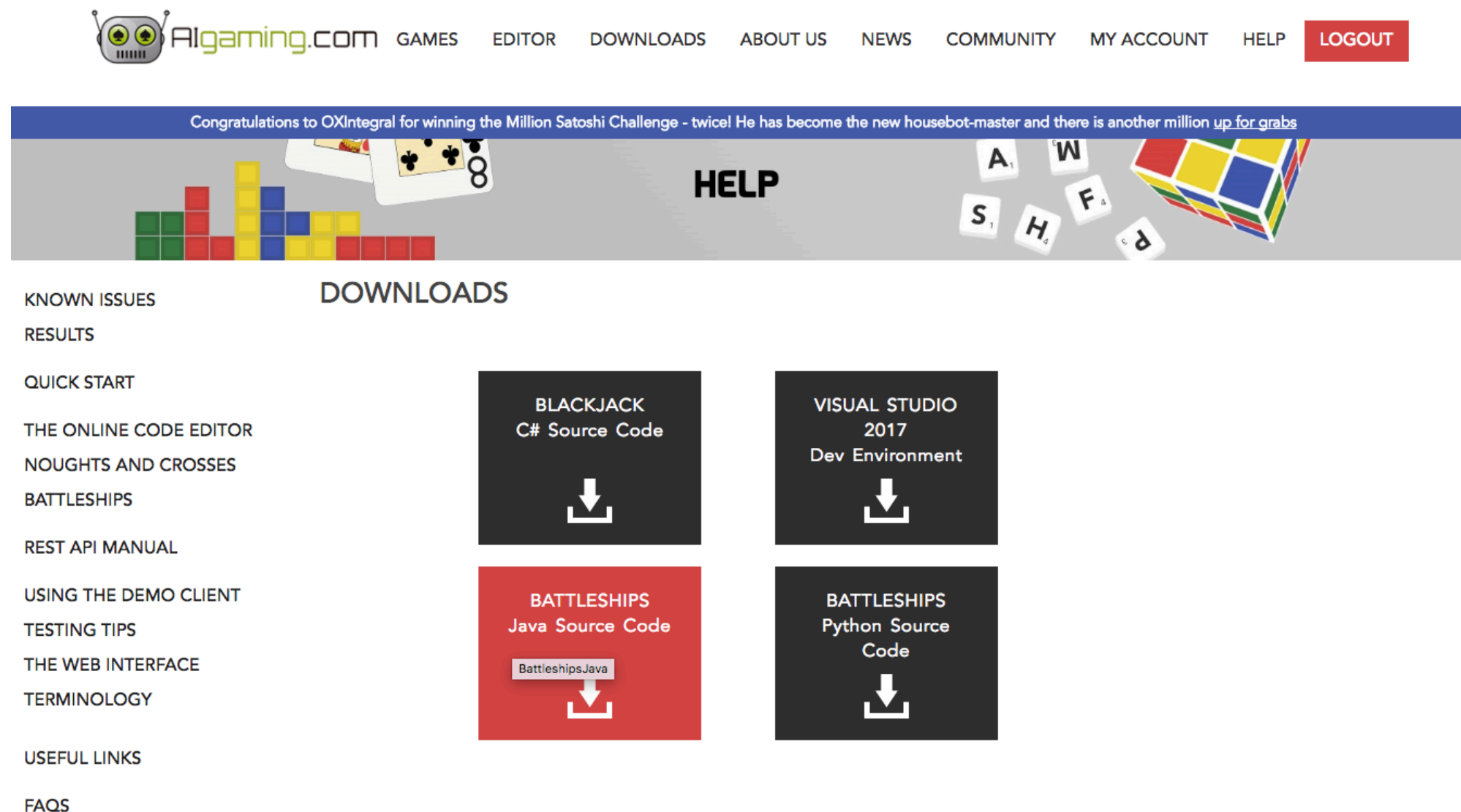
- **Problem:** in the online editor you cannot *memorize* the previous search state, in particular, your previous moves.
- Hence, you do not know where to *backtrack*.
- **Solution:** go to the source code!





# Using the demo client

<https://www.aigaming.com/Help?url=downloads>



The screenshot shows the Aigaming.com website. The top navigation bar includes the Aigaming.com logo and links for GAMES, EDITOR, DOWNLOADS, ABOUT US, NEWS, COMMUNITY, MY ACCOUNT, HELP, and a red LOGOUT button. A blue banner below the navigation bar reads: "Congratulations to OXIntegral for winning the Million Satoshi Challenge - twice! He has become the new housebot-master and there is another million up for grabs". The main content area has a grey background with the word "HELP" in large black letters. Below "HELP" is a "DOWNLOADS" section with four download buttons: "BLACKJACK C# Source Code", "VISUAL STUDIO 2017 Dev Environment", "BATTLESHIPS Java Source Code" (highlighted in red), and "BATTLESHIPS Python Source Code". Each button has a white download icon. On the left side of the page, there is a vertical list of links: KNOWN ISSUES, RESULTS, QUICK START, THE ONLINE CODE EDITOR, NOUGHTS AND CROSSES, BATTLESHIPS, REST API MANUAL, USING THE DEMO CLIENT, TESTING TIPS, THE WEB INTERFACE, TERMINOLOGY, USEFUL LINKS, and FAQs.

KNOWN ISSUES

RESULTS

QUICK START

THE ONLINE CODE EDITOR

NOUGHTS AND CROSSES

BATTLESHIPS

REST API MANUAL

USING THE DEMO CLIENT

TESTING TIPS

THE WEB INTERFACE

TERMINOLOGY

USEFUL LINKS

FAQS

DOWNLOADS

BLACKJACK  
C# Source Code

↓

VISUAL STUDIO  
2017  
Dev Environment

↓

BATTLESHIPS  
Java Source Code

BattleshipsJava

↓

BATTLESHIPS  
Python Source  
Code

↓

# The important parts

- ▶ **battleships\_layout.py:**

- ▶ function *play\_game*

```
474  
475 def play_game(self):  
476     """Play a game."""  
477     self.resultText.config(text='Playing game')  
478     self.in_game = True  
479
```

- ▶ the *while* loop

```
502  
503  
504  
505  
506  
507  
508 while True:  
509     if self.game_cancelled:  
510         break  
  
511     if game_state['IsMover']:  
512         self.resultText.config(text='Playing Game - Your Turn')  
513         move = battleships_move.calculateMove(game_state)  
514         move_results = self.make_move(move)
```

**Here you can store your moves!**



# The important parts

- ▶ **battleships\_move.py:**

- ▶ function *calculateMove*

```
4
5 def calculateMove(gamestate):
6     if gamestate["Round"] == 0: # If we are in the ship placement round
7         # move = exampleShipPlacement() # Does not take land into account
8         move = deployRandomly(gamestate) # Randomly place your ships
9     else: # If we are in the ship hunting round
10        move = chooseRandomValidTarget(gamestate) # Randomly fire at valid sea targets
11    return move
12
```

**Here you can modify your moves strategy and *search*!**



**Homework: integrate DFS with pruning strategy into the battleships code**

