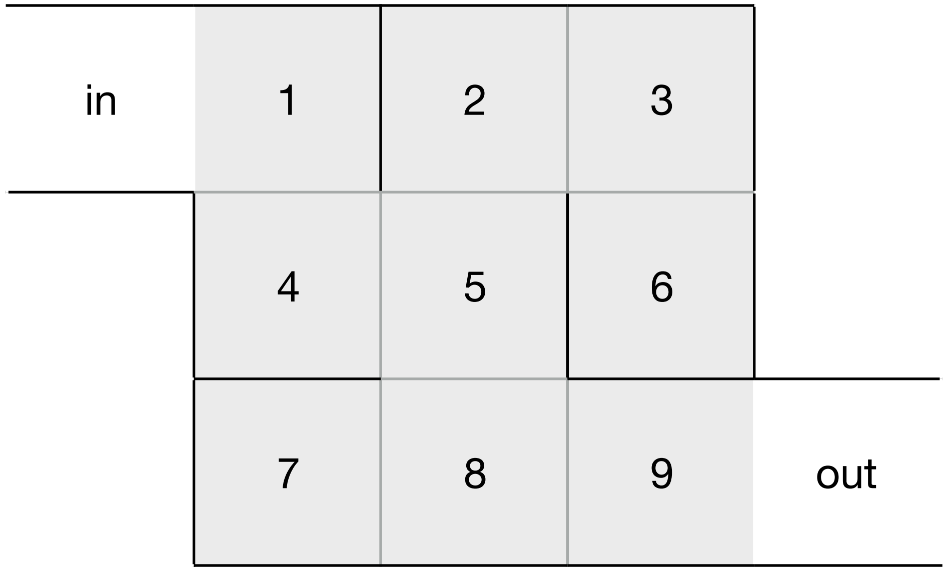
AI Games course

Certificate 1, session 1

The maze example



in

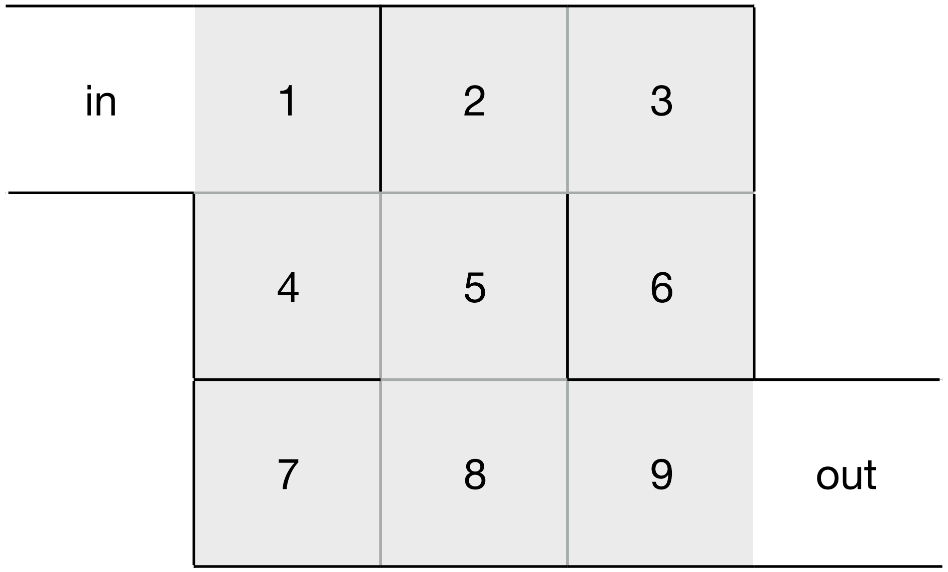
1 2 3

4 5 6

7 8 9

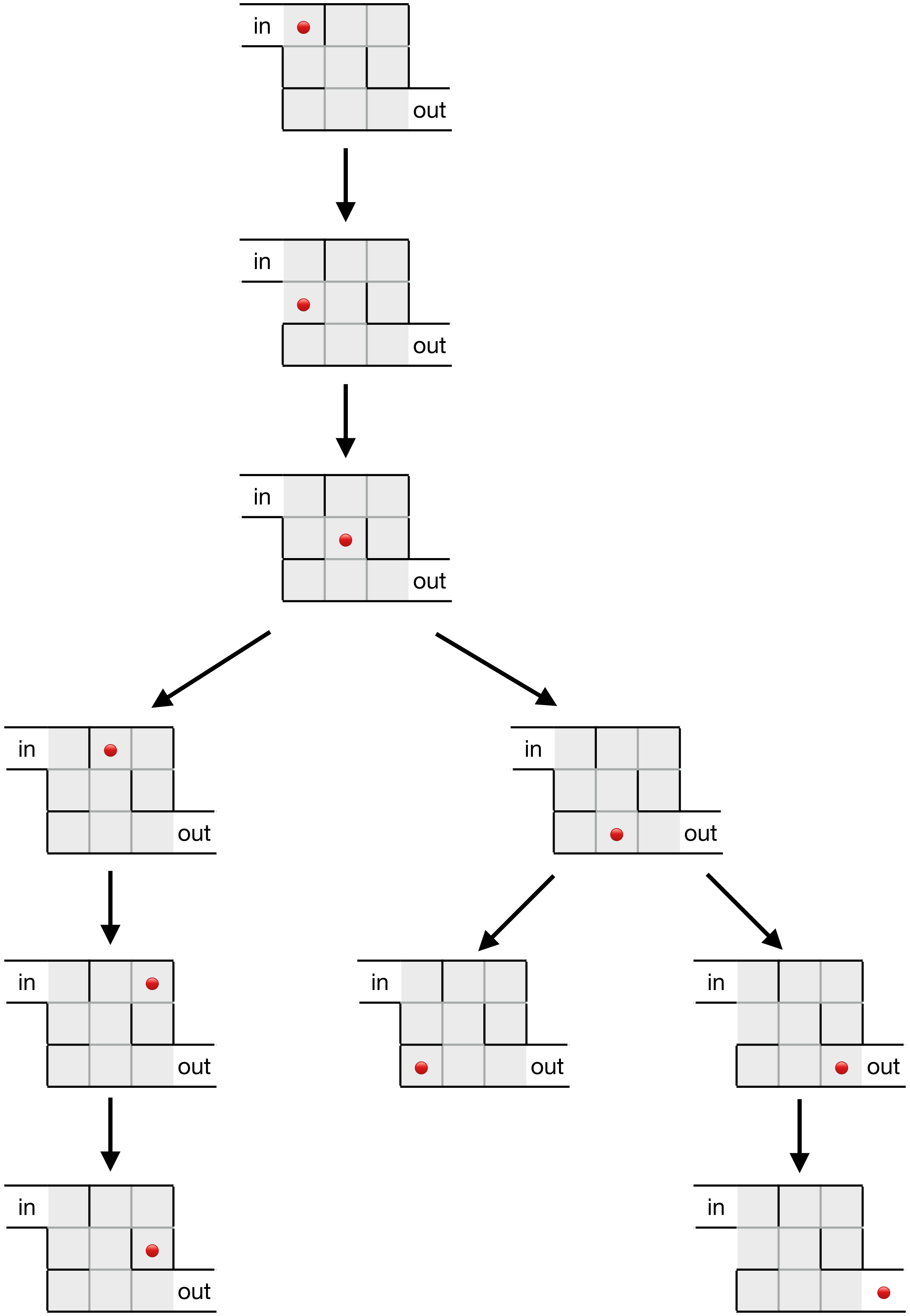
out

The maze example: graph representation

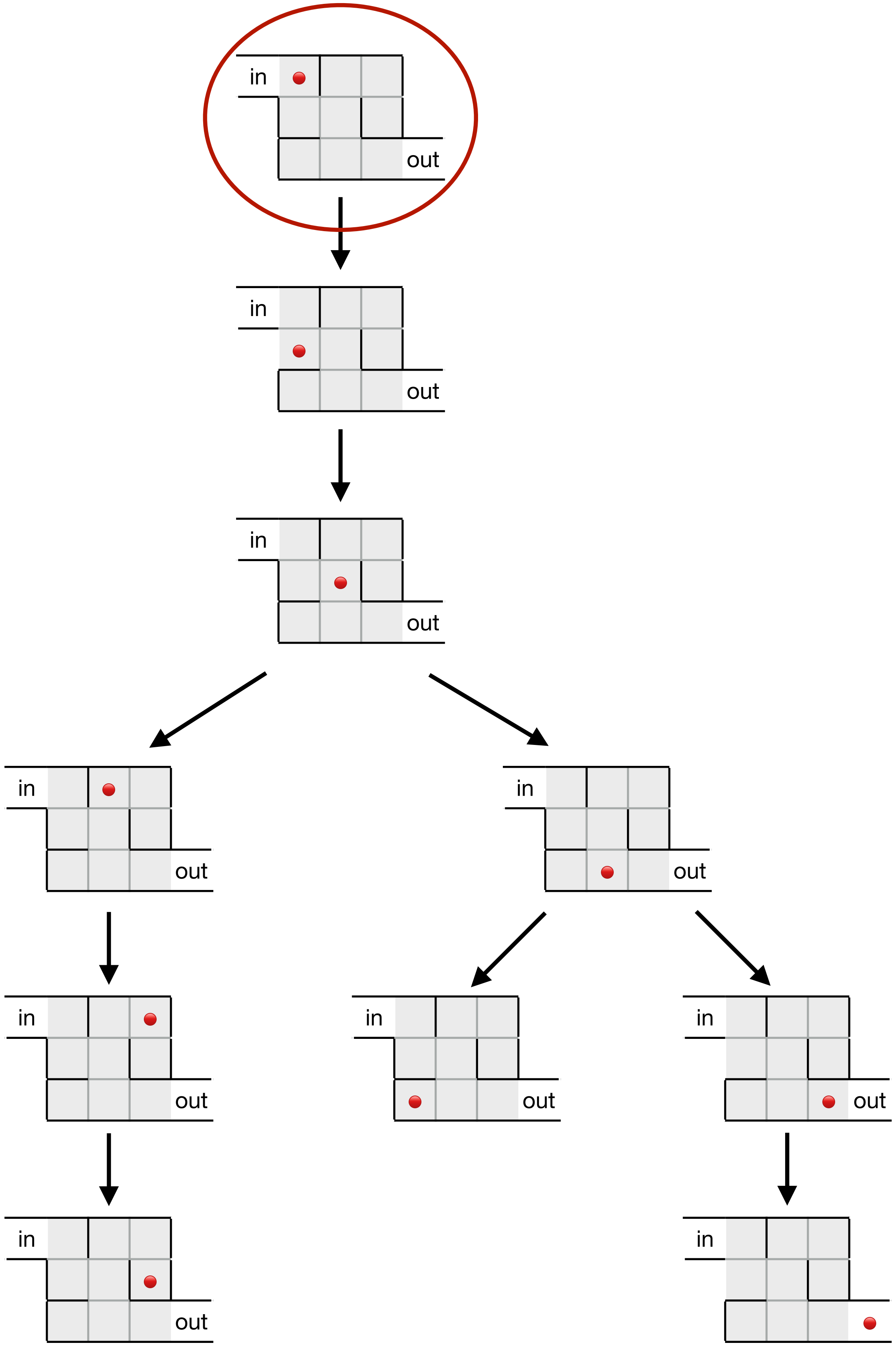


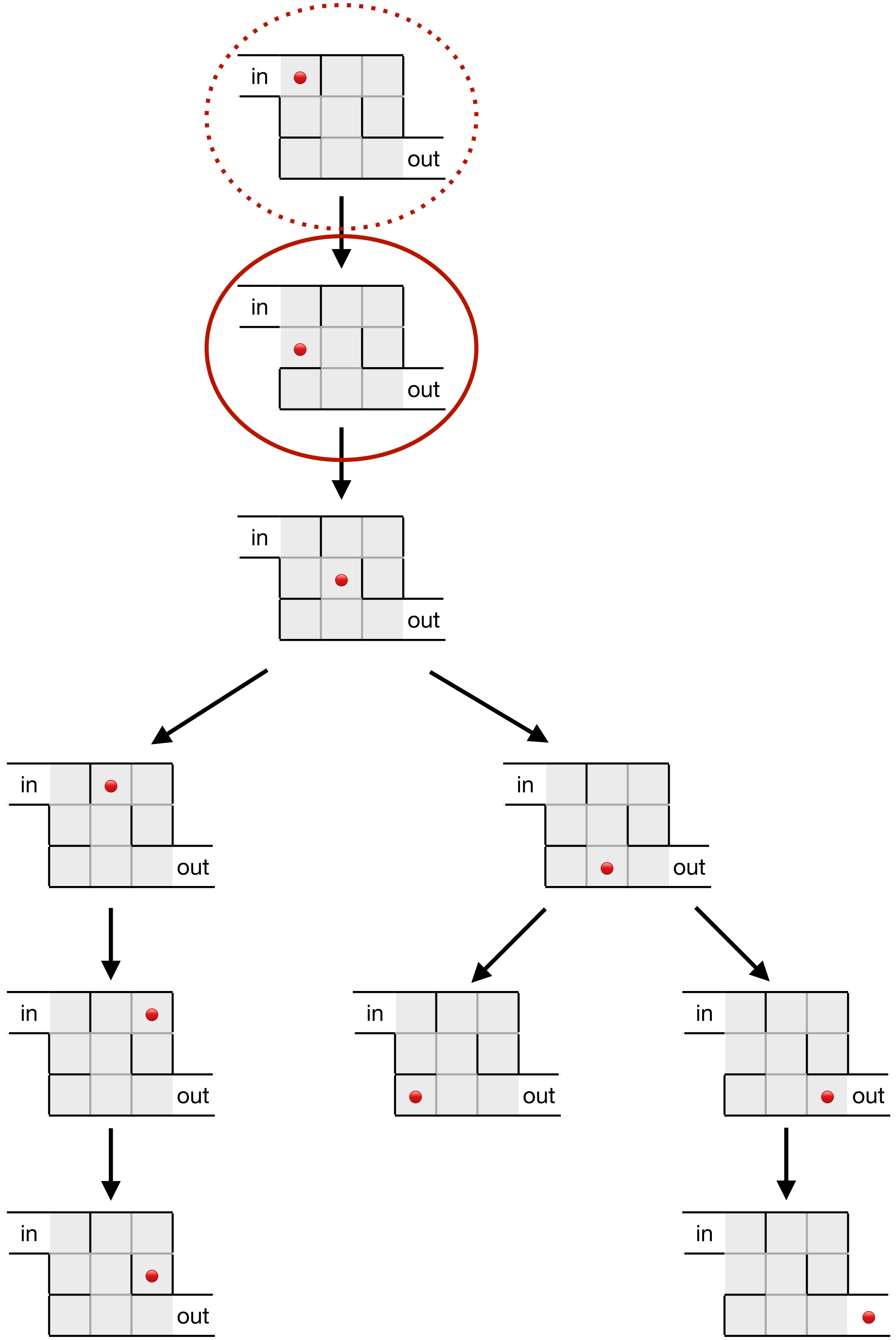
```
maze = {'in': {1},
        1: {’in’,4},
        2: {3,5},
        3: {2,6},
        4: {1,5},
        5: {2,4,8},
        6: {3},
        7: {8},
        8: {5,7,9},
        9: {8,’out'},
        'out': {9}}
```

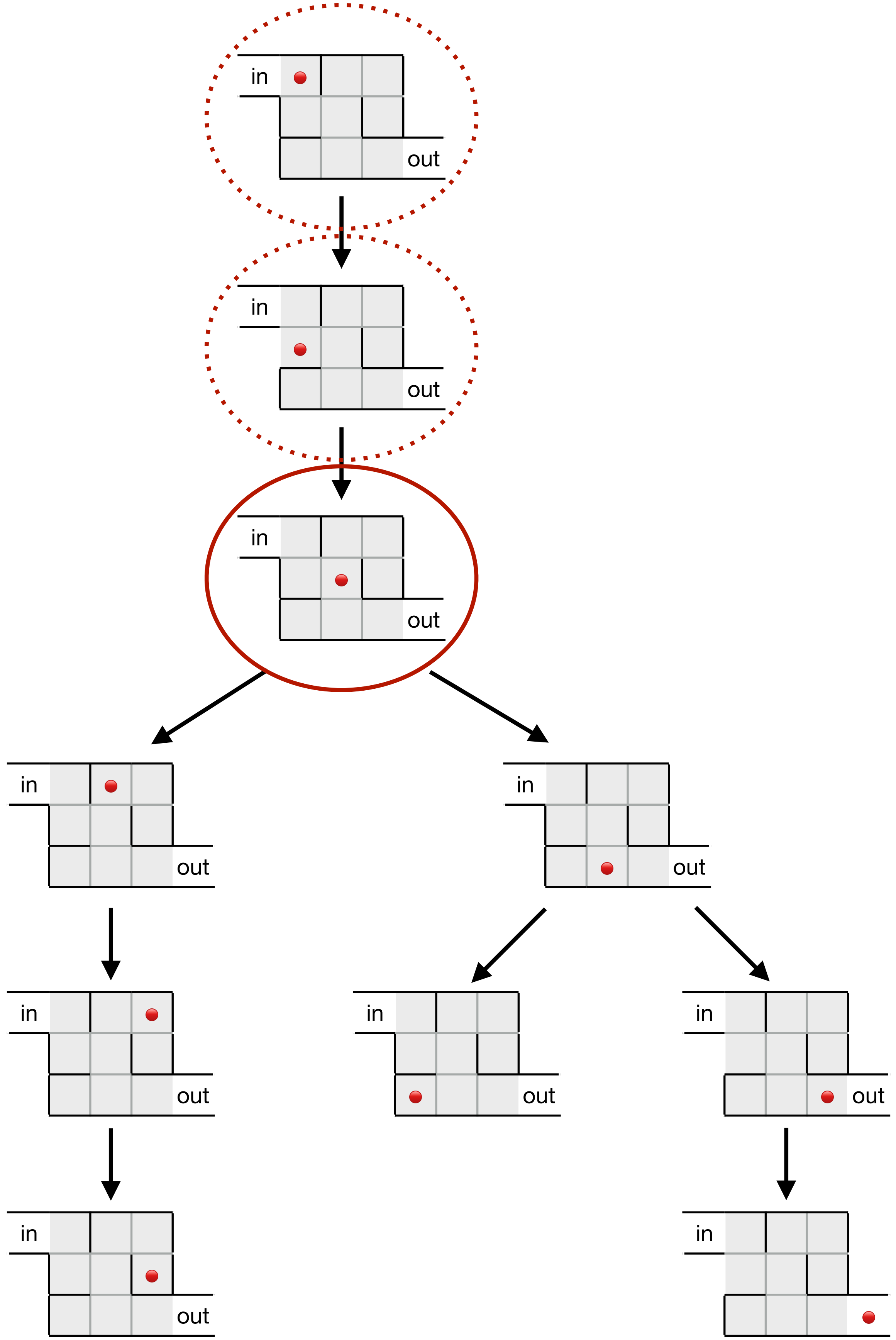
Maze
“space tree”

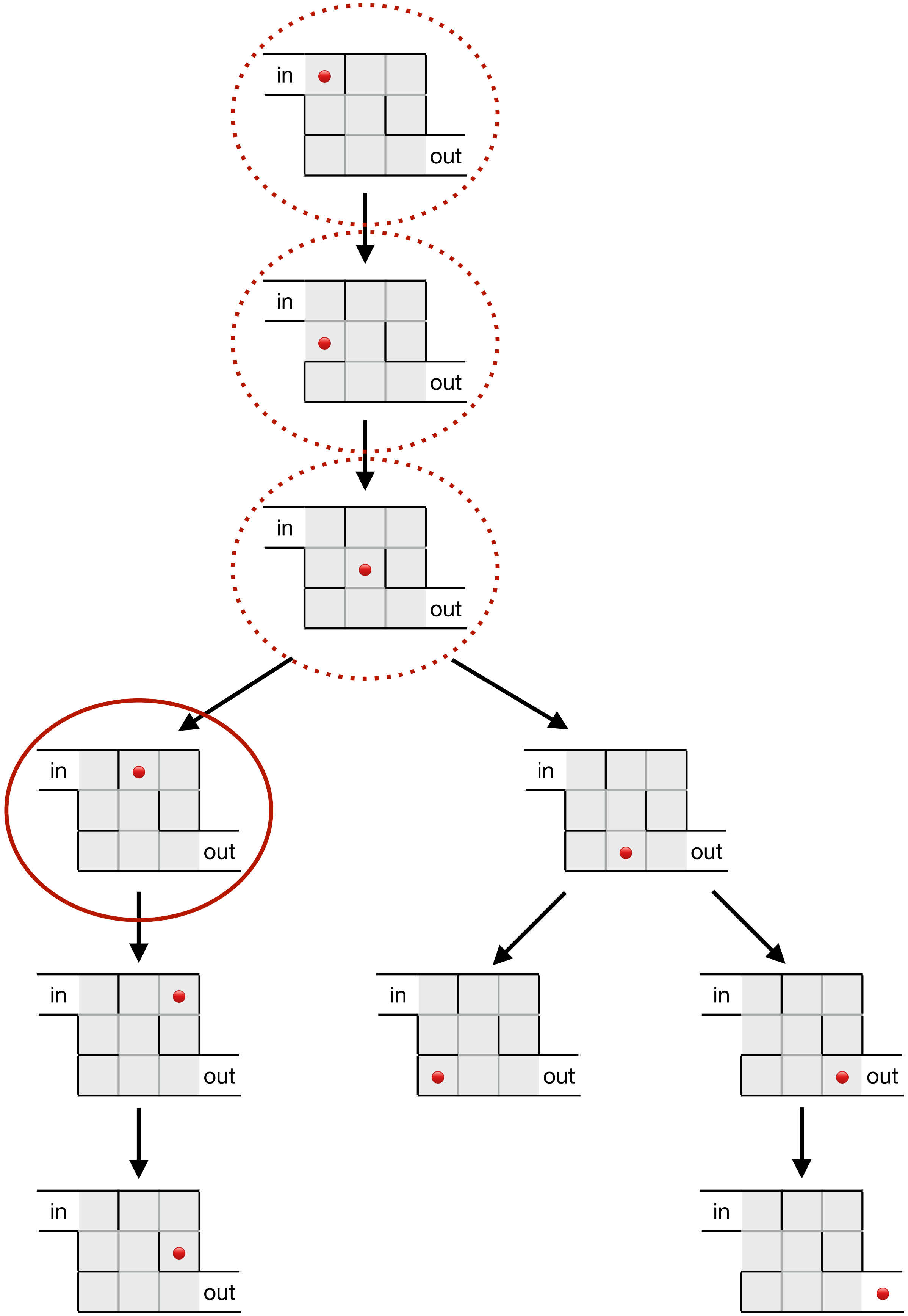


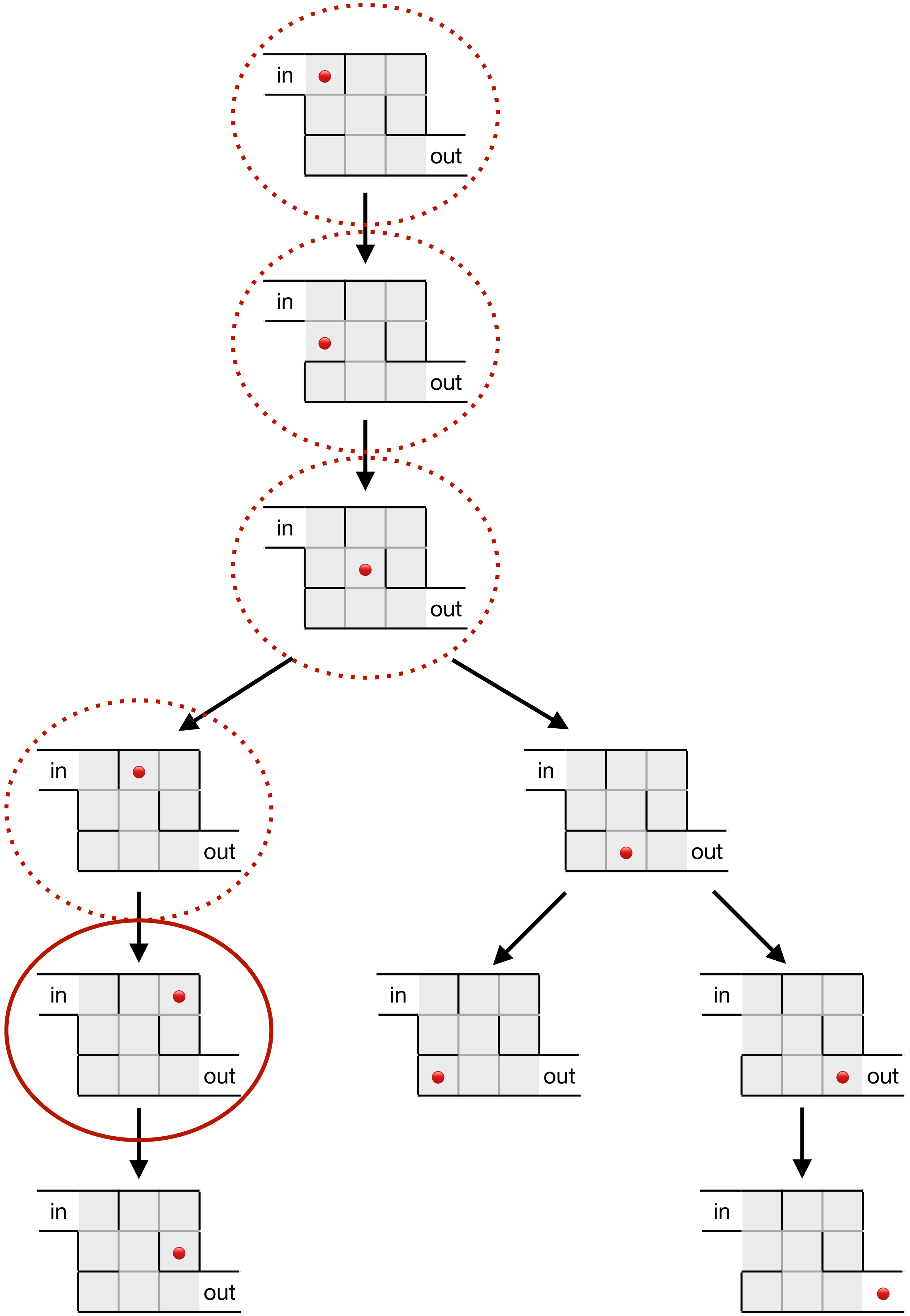
Depth-first search

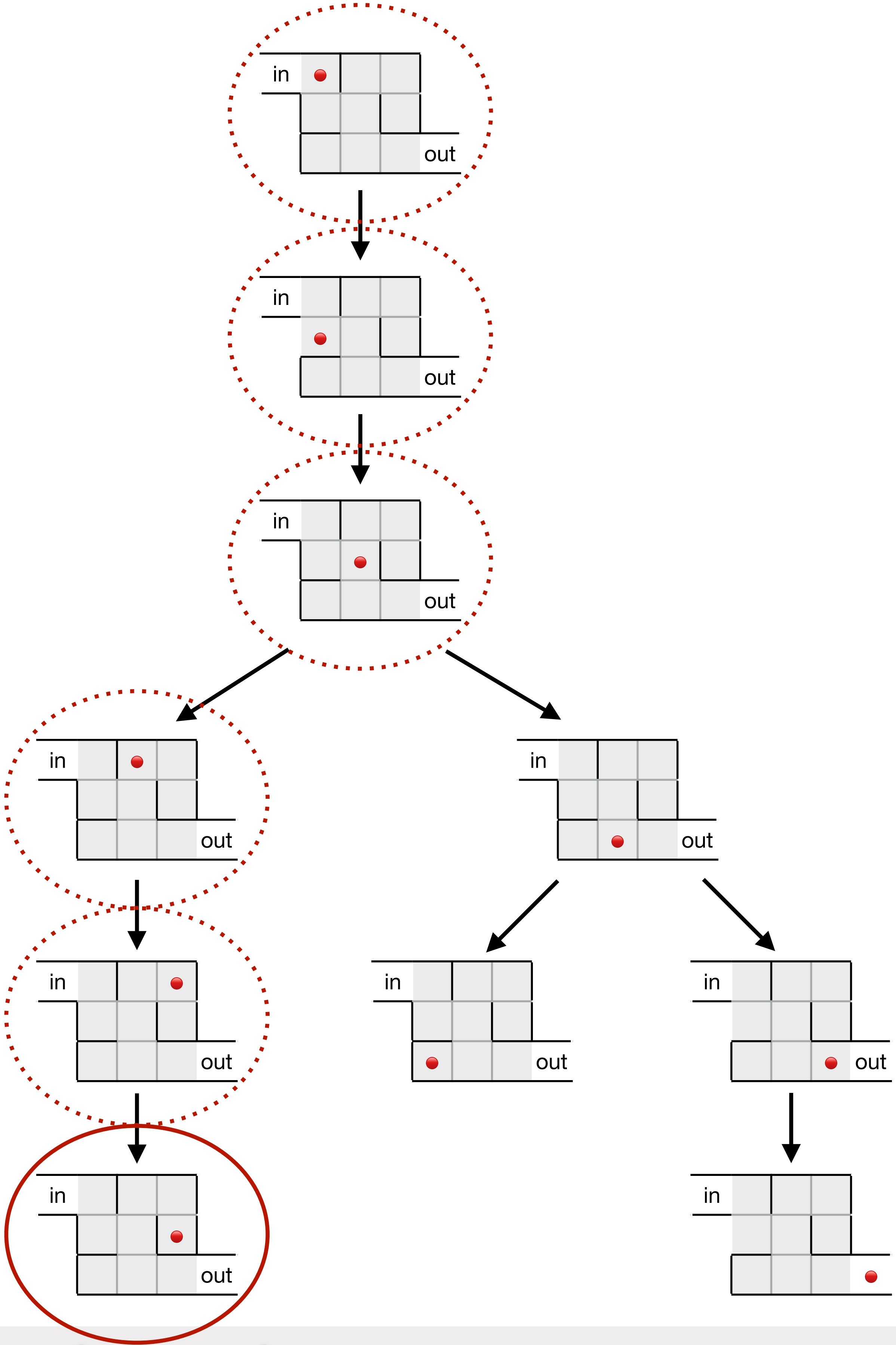

in
out

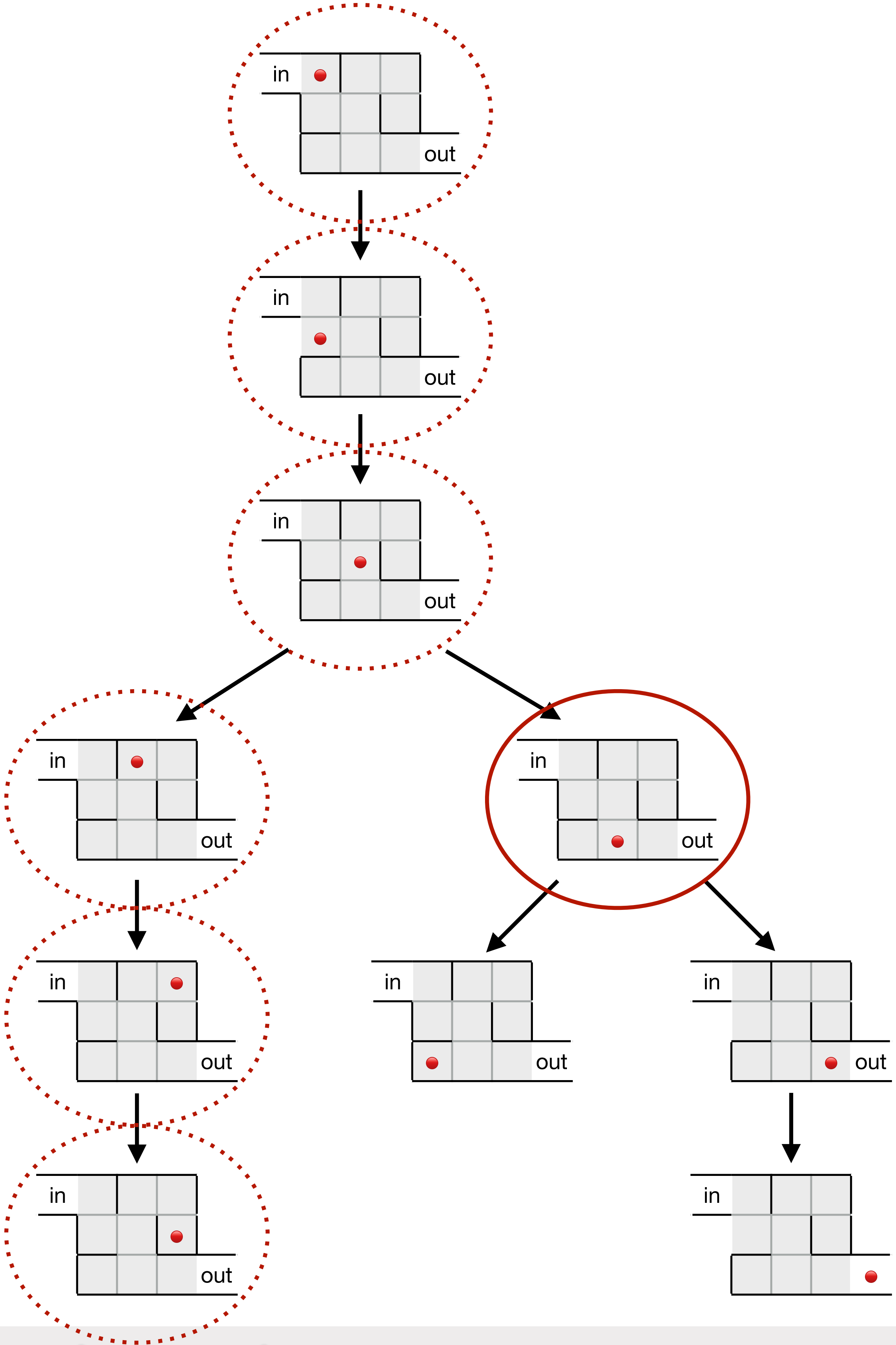

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in

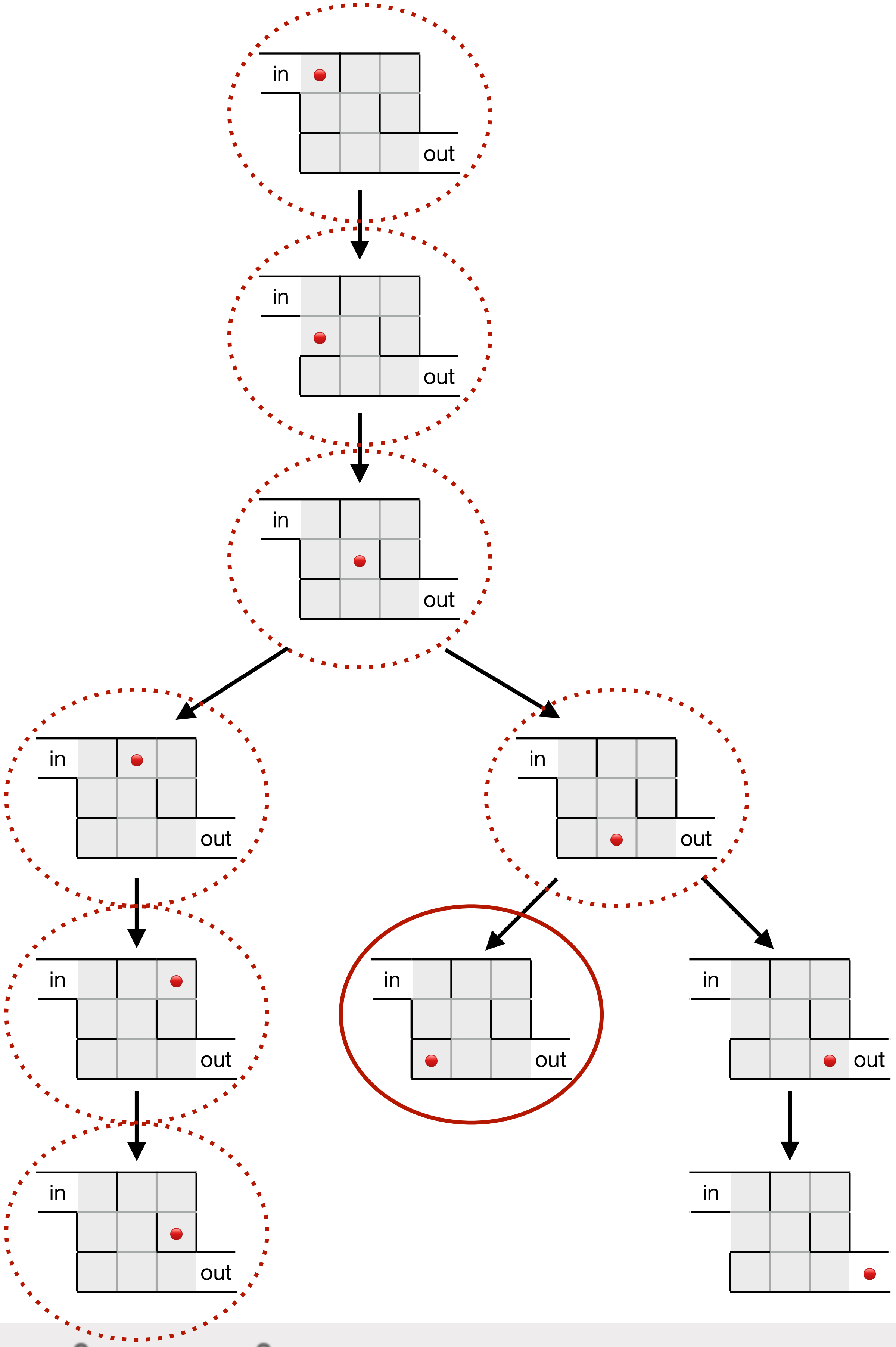

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in

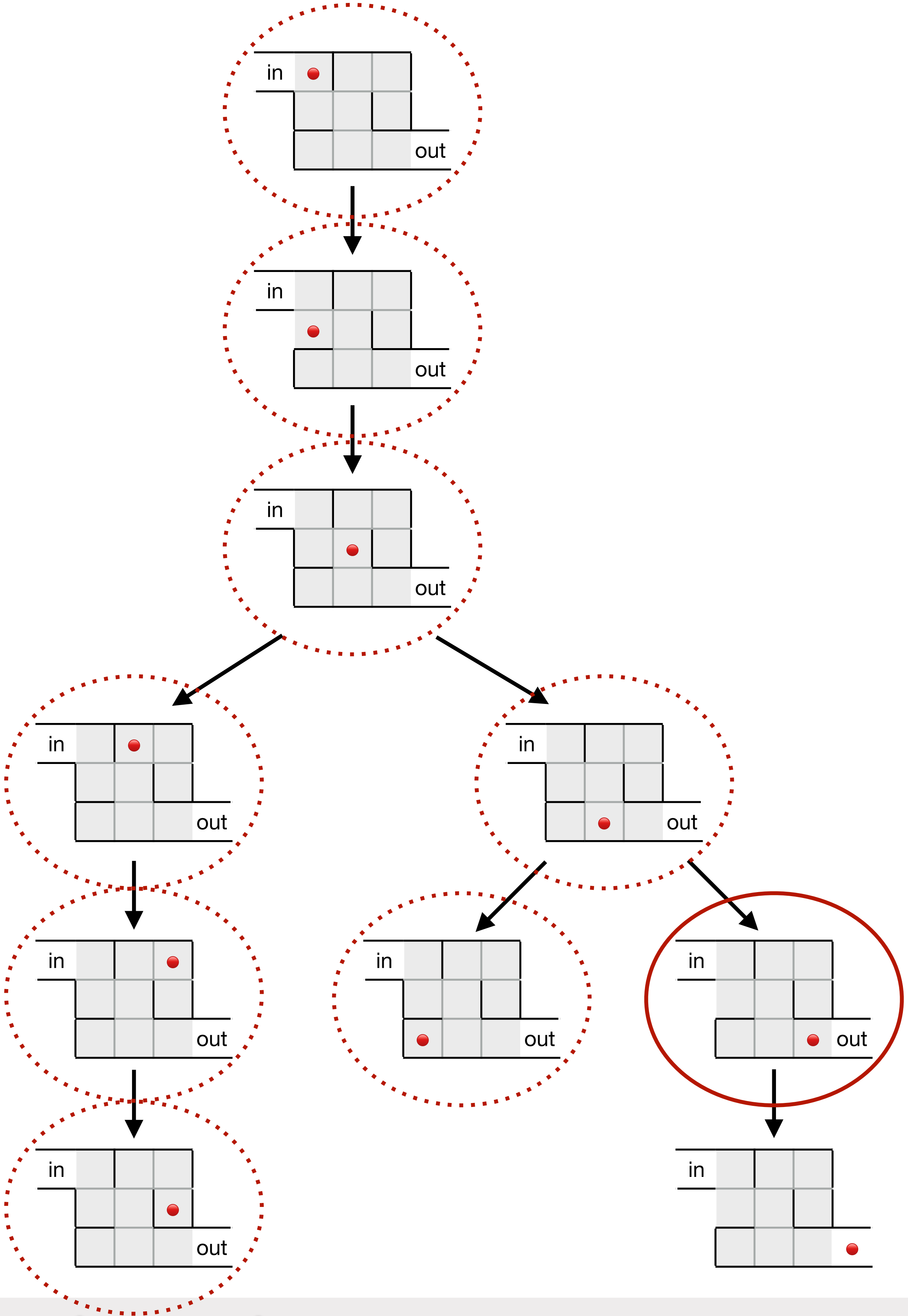

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
in

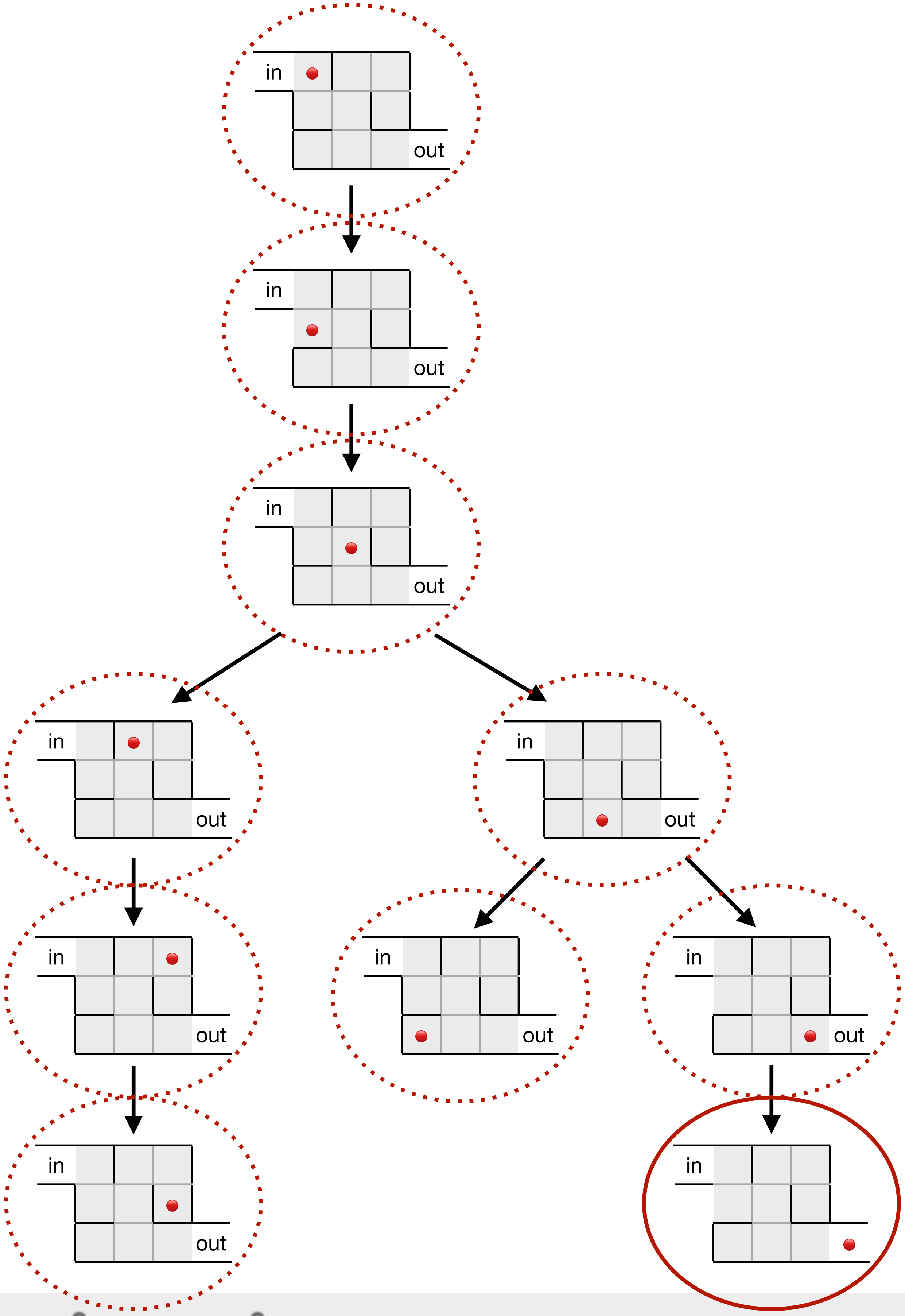

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
in


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in


in
out


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in

DFS algorithm

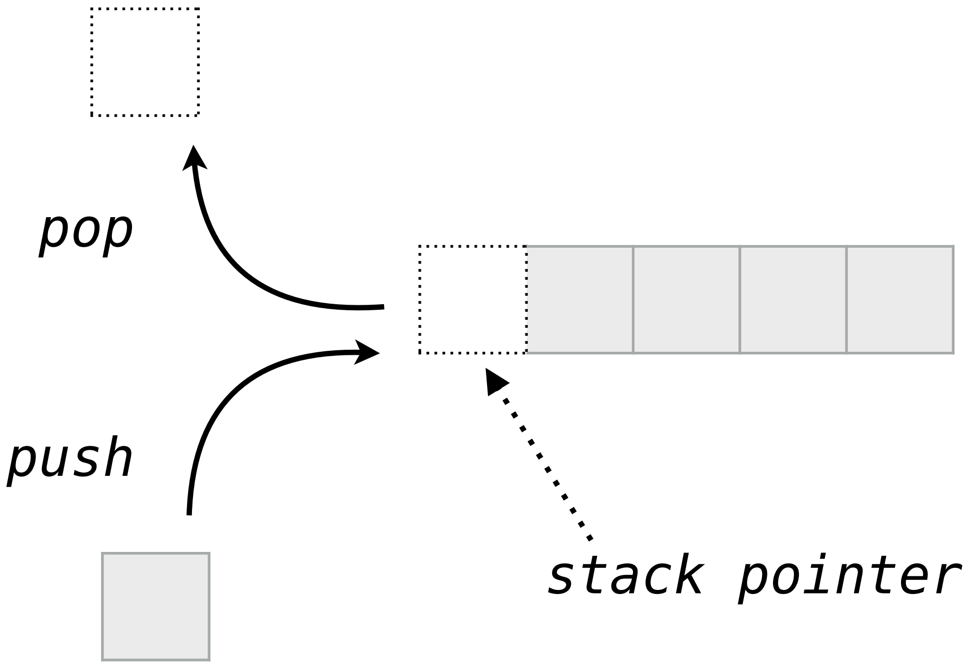
DFS algorithm

```
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 into the stack all the nodes reachable from the top node and
   not yet visited
   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 into the stack all the nodes reachable from the top node and
   not yet visited
   3. while the stack is not empty, recursively apply steps 1-3
```

Stack:



LIFO:
last in,
first out

DFS algorithm

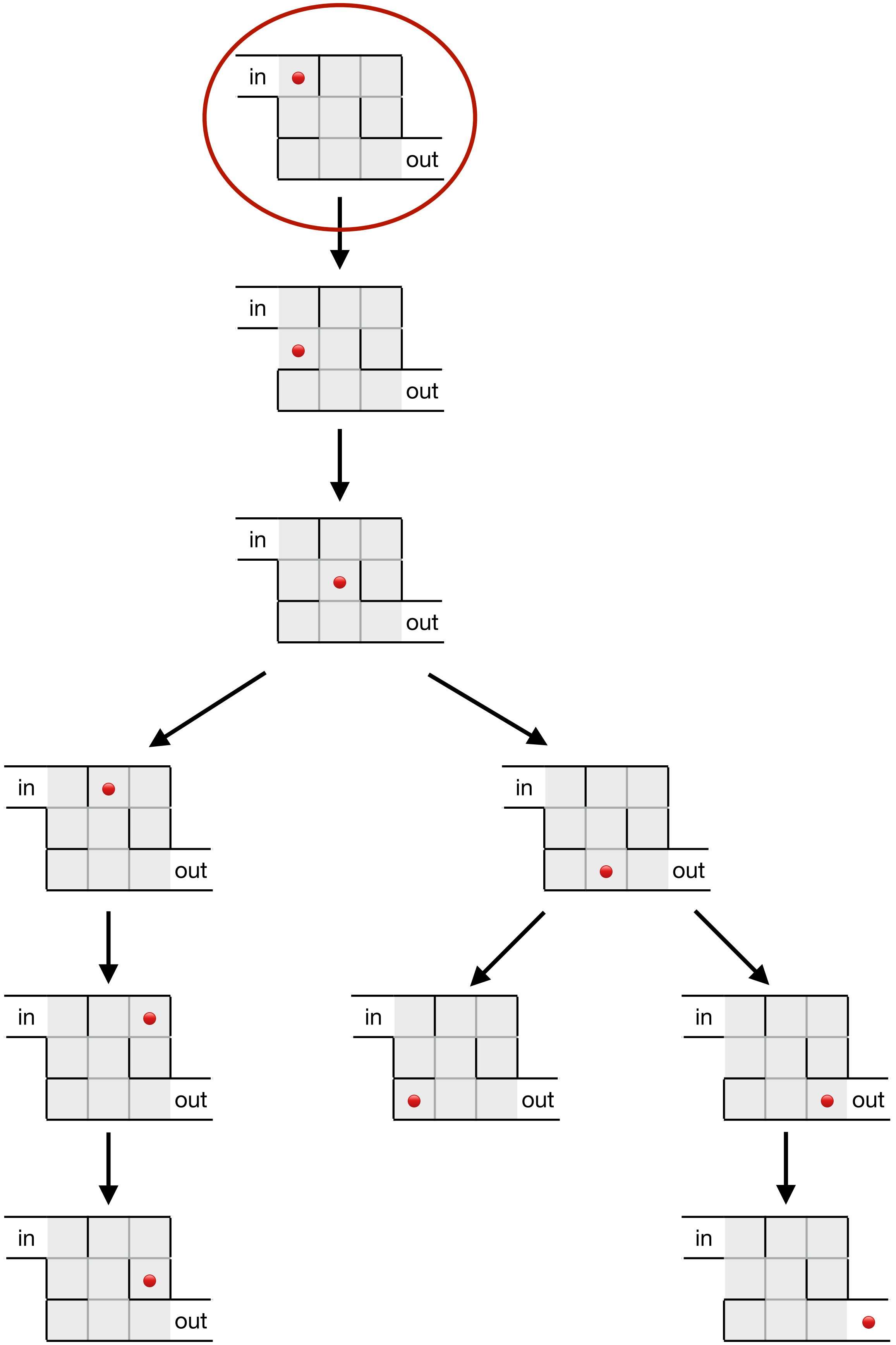
```
def dfs(graph, start):
    visited, stack = set(), [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)

            print(vertex)
```

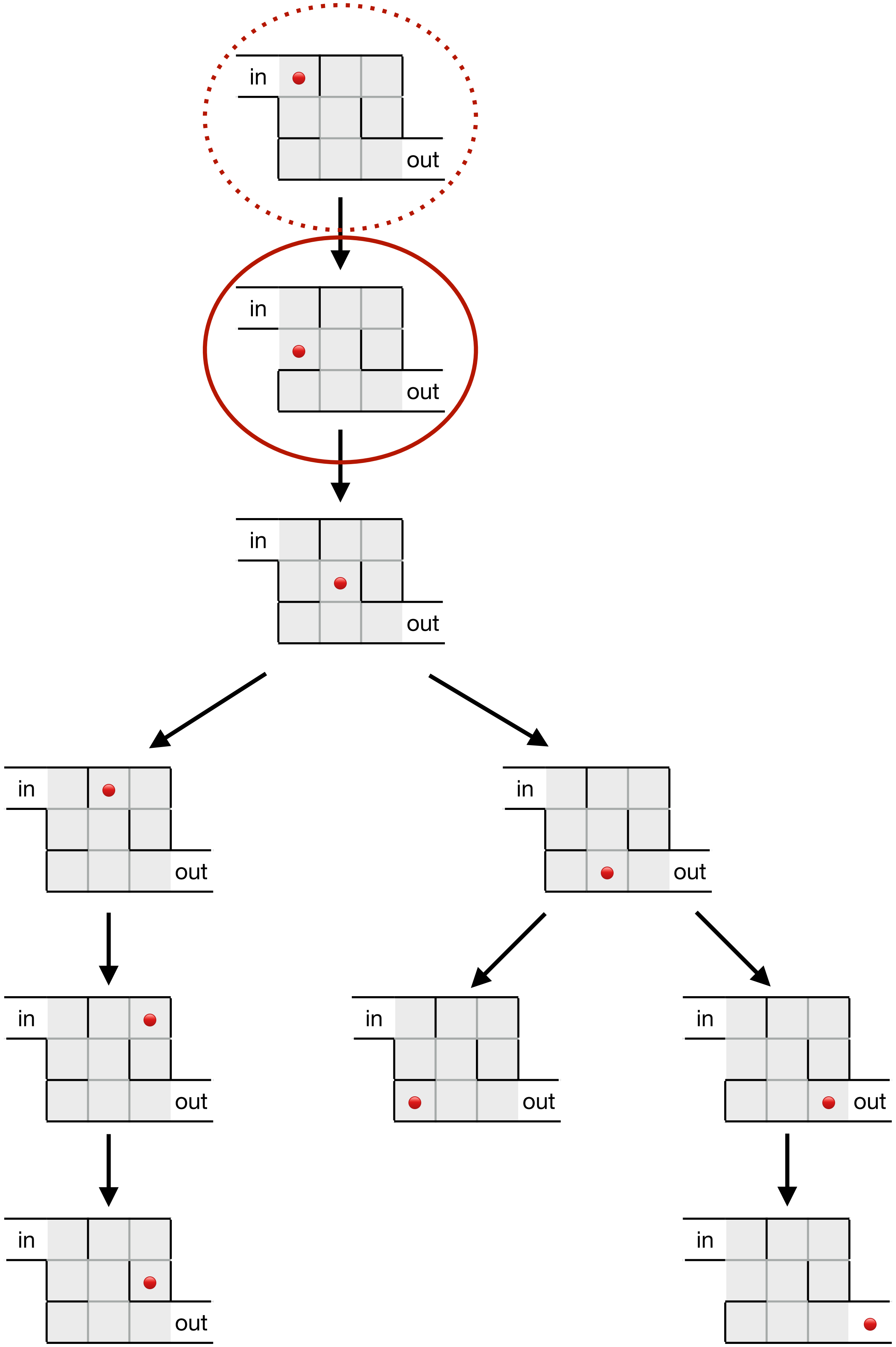
DFS algorithm

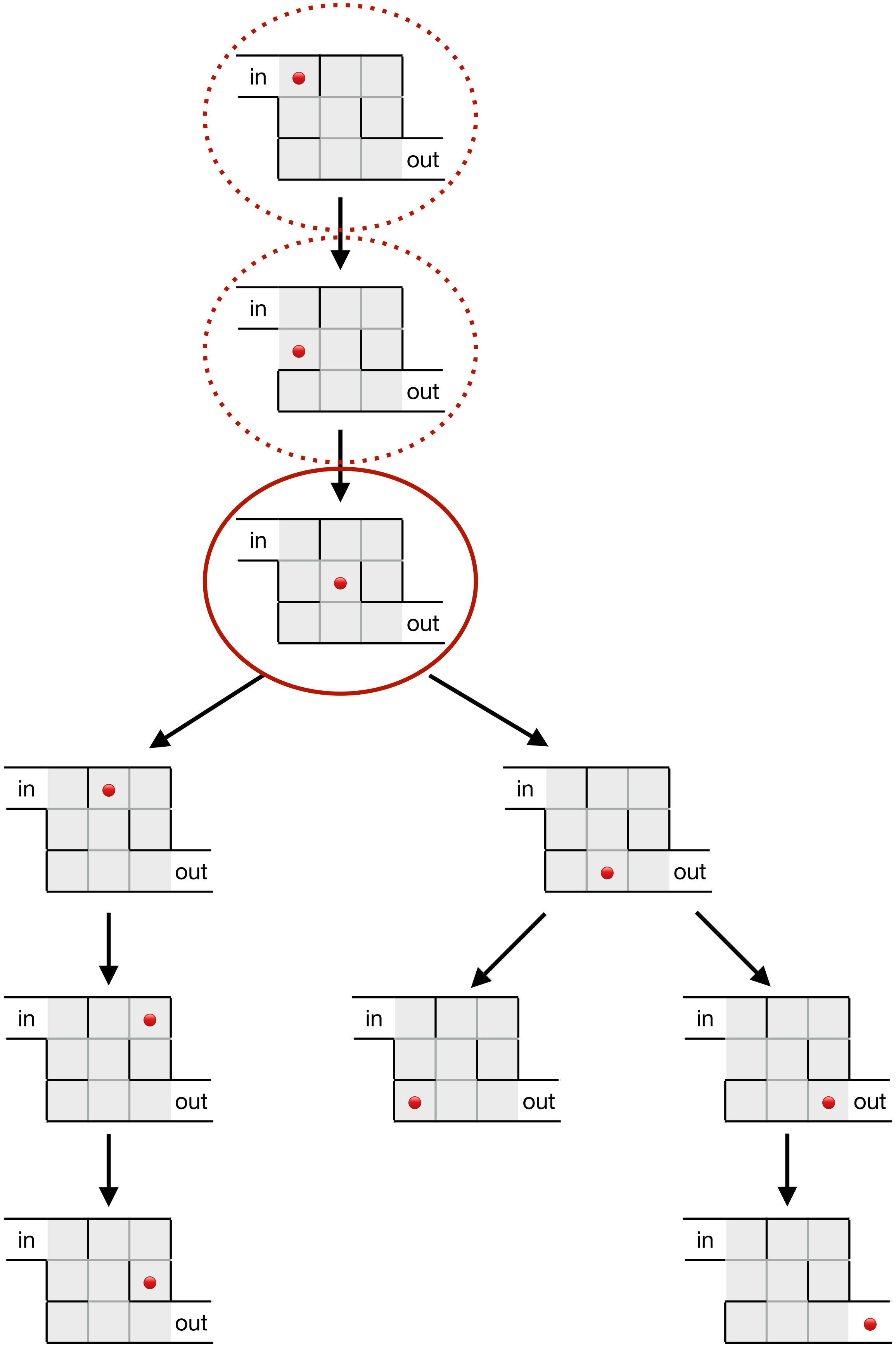
```python
def dfs(graph, start):
    visited, stack = set(), [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
           visited.add(vertex)
           stack.extend(graph[vertex] - visited)
           print(vertex)
```

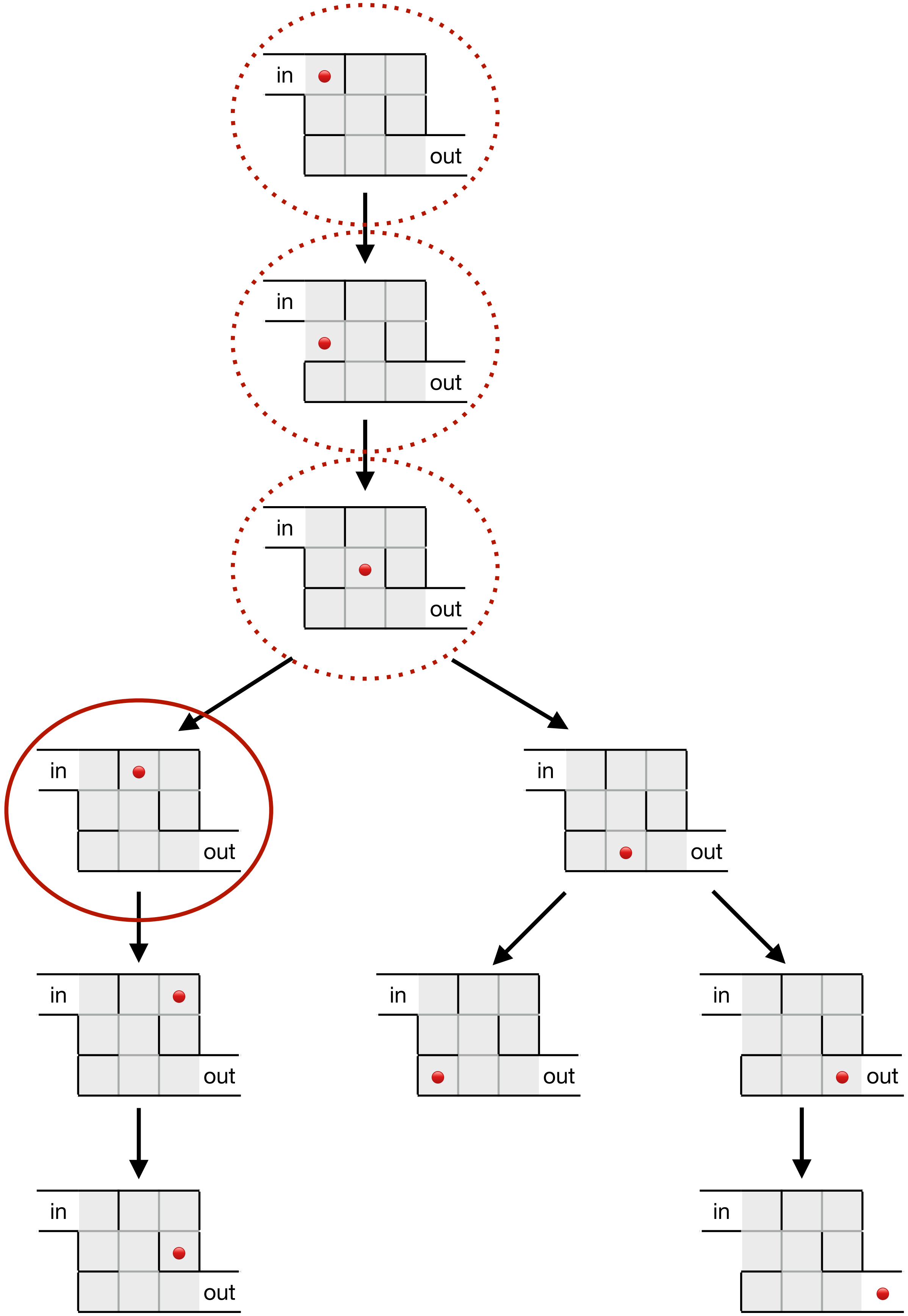
```
>>> dfs(maze,'in')
```

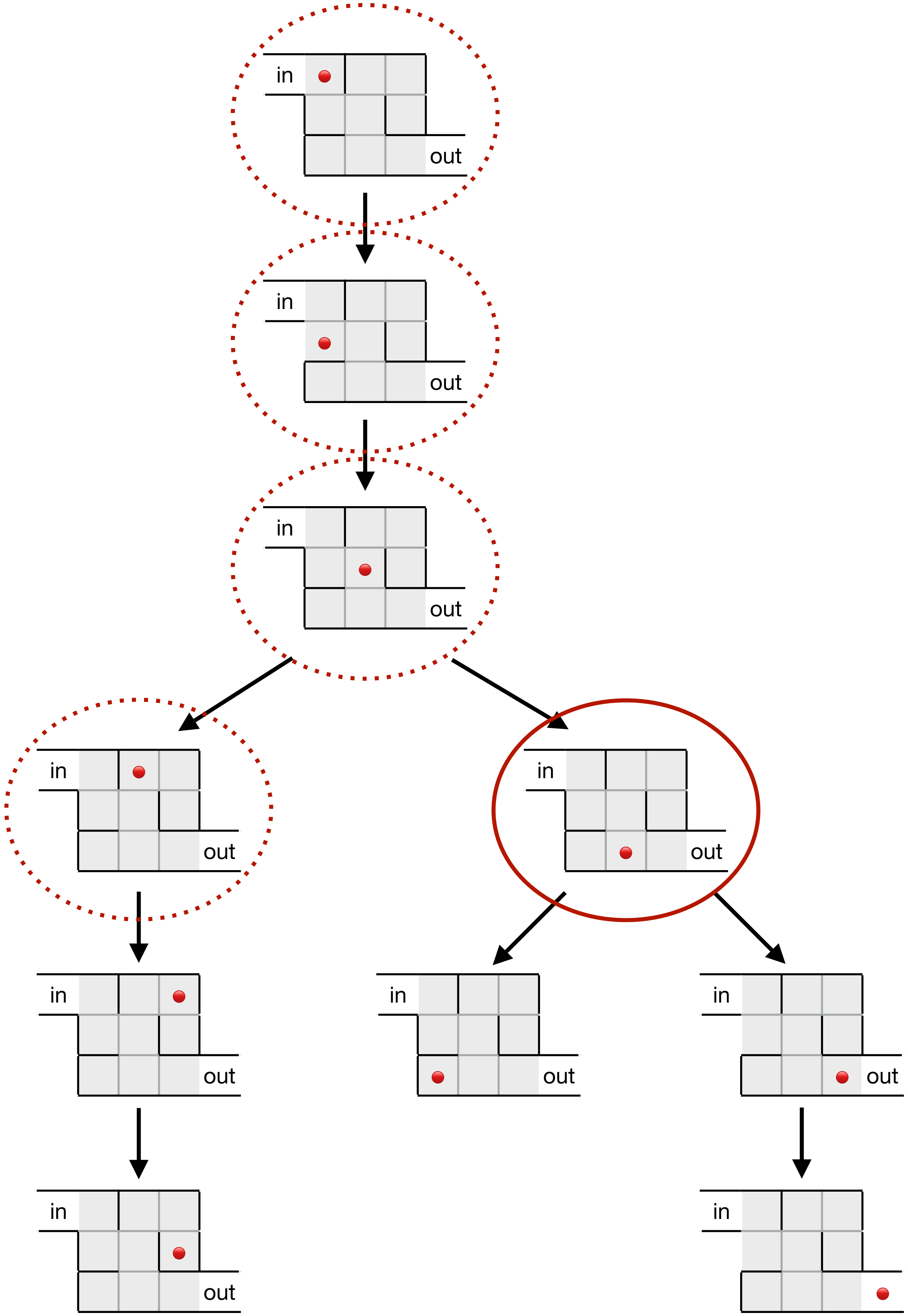
Breadth-first search

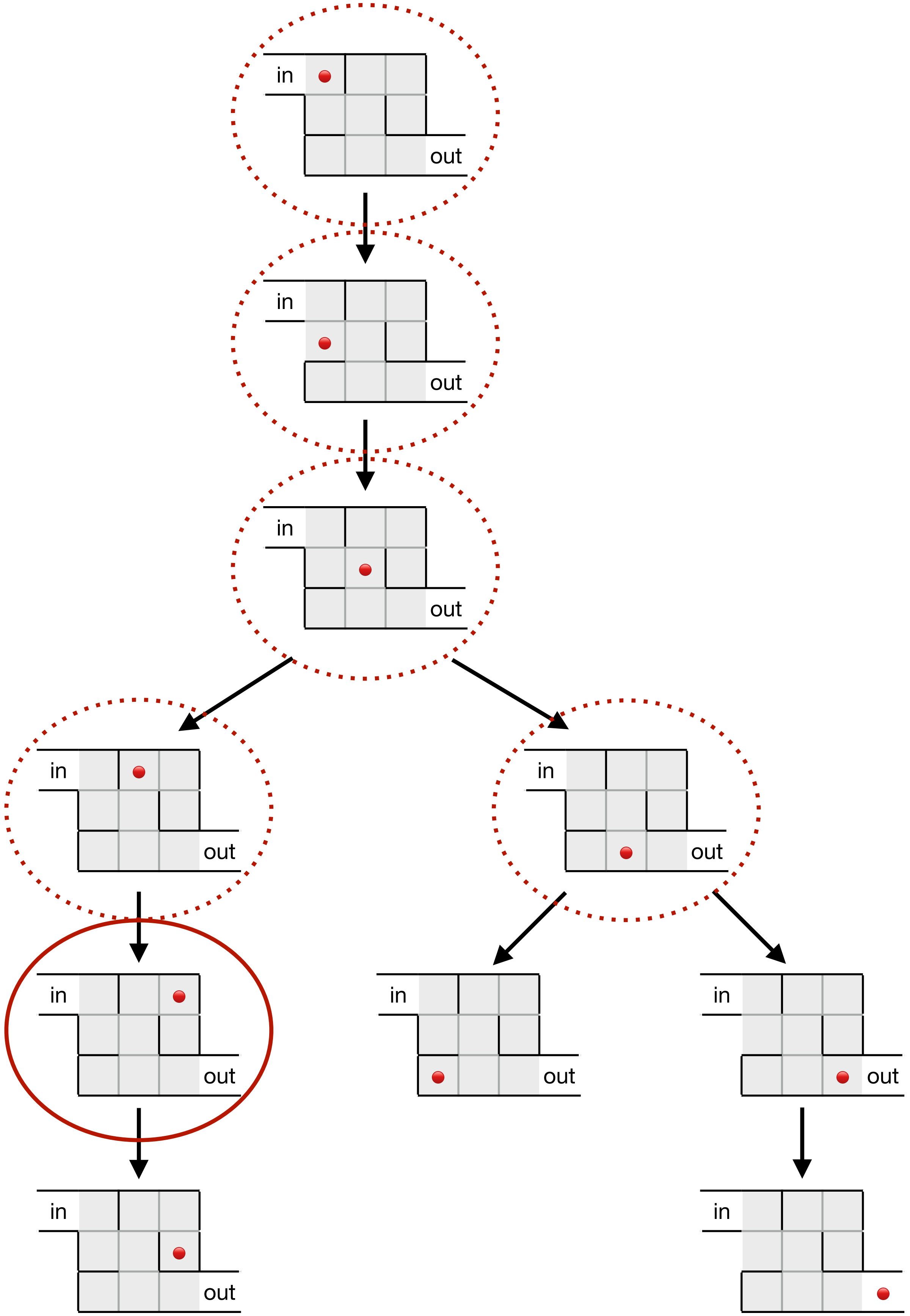

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in

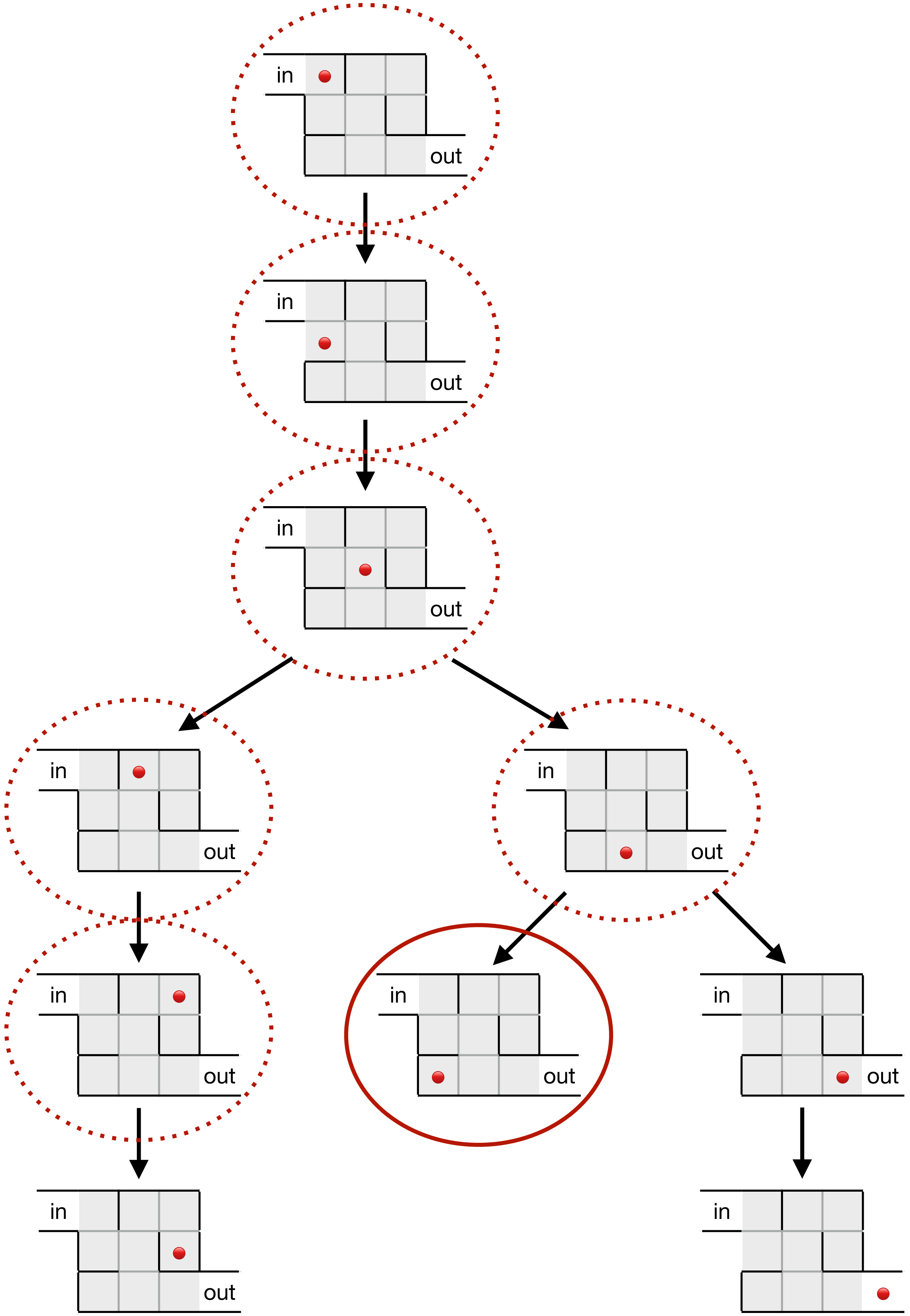

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in

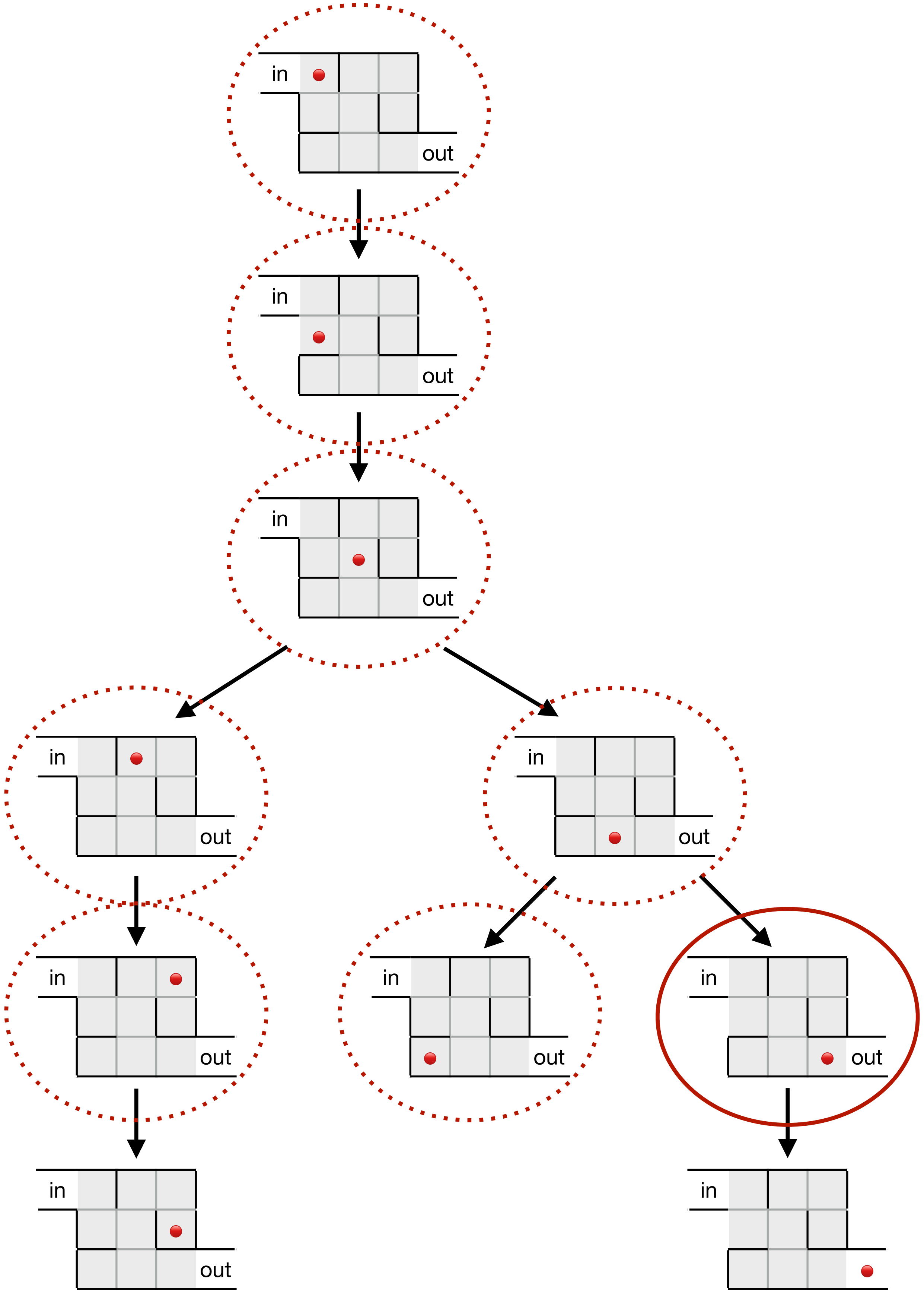

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in

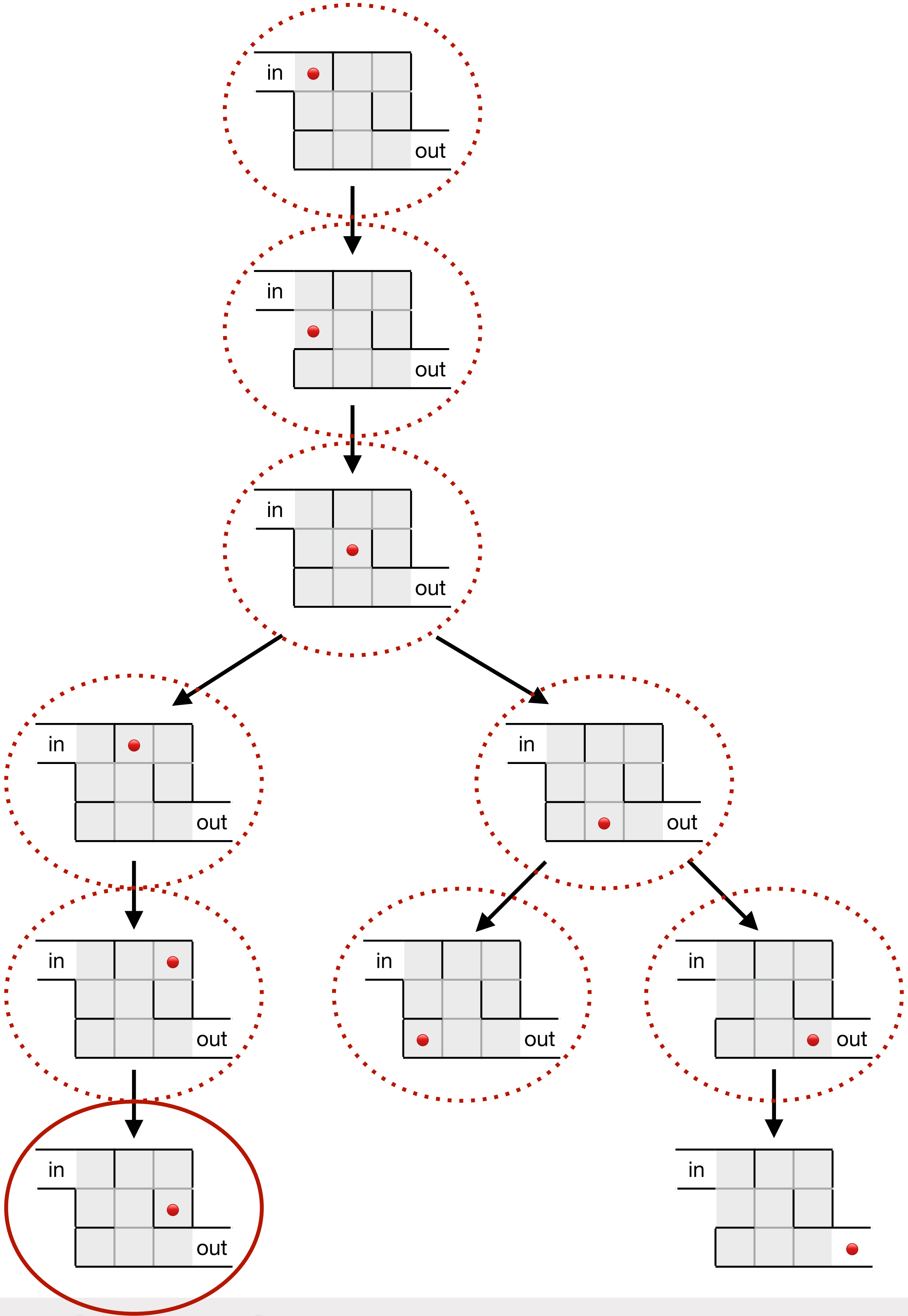

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in

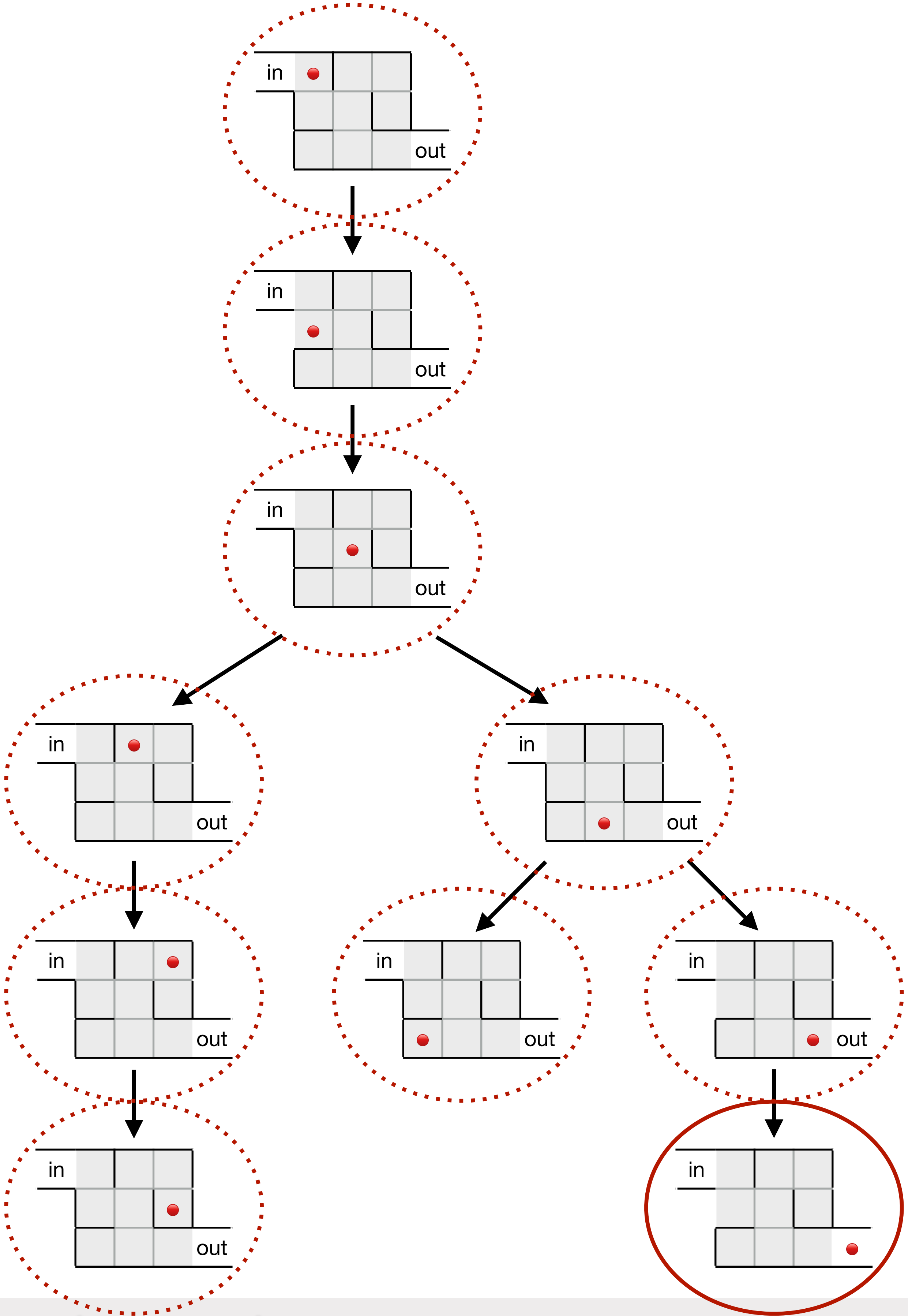

in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in


in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in
out
in

BFS algorithm

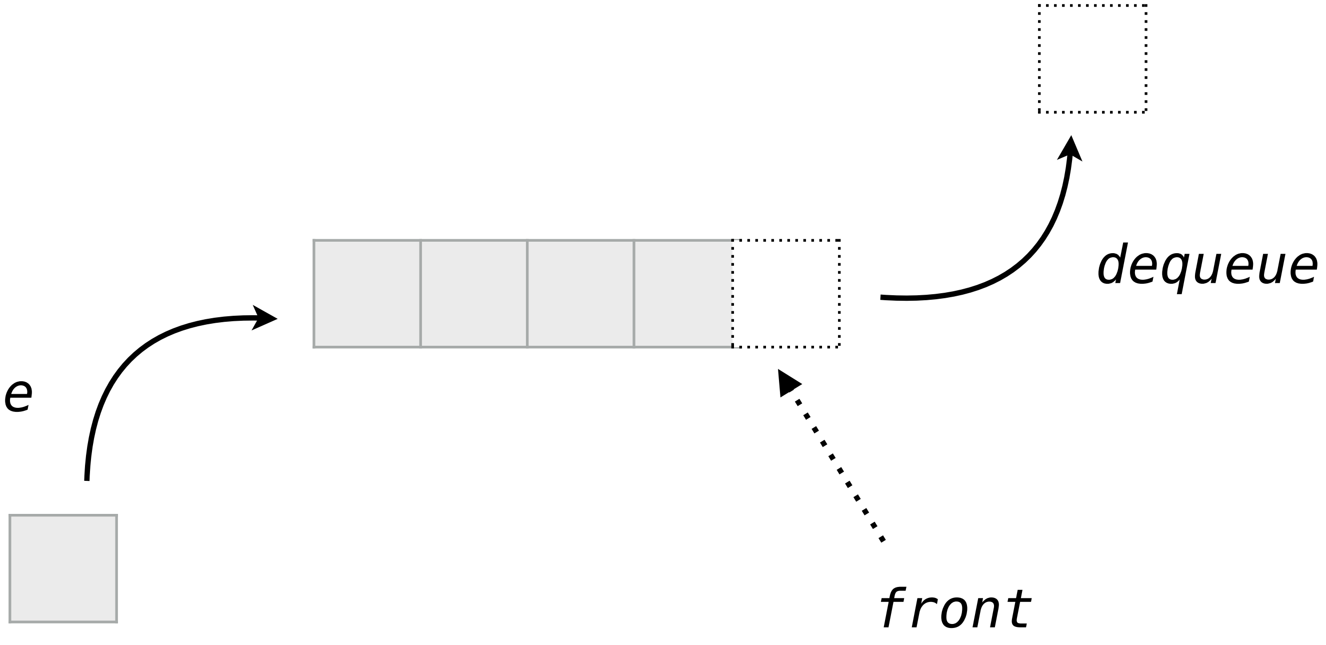
BFS algorithm

```
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

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

            print(vertex)
```

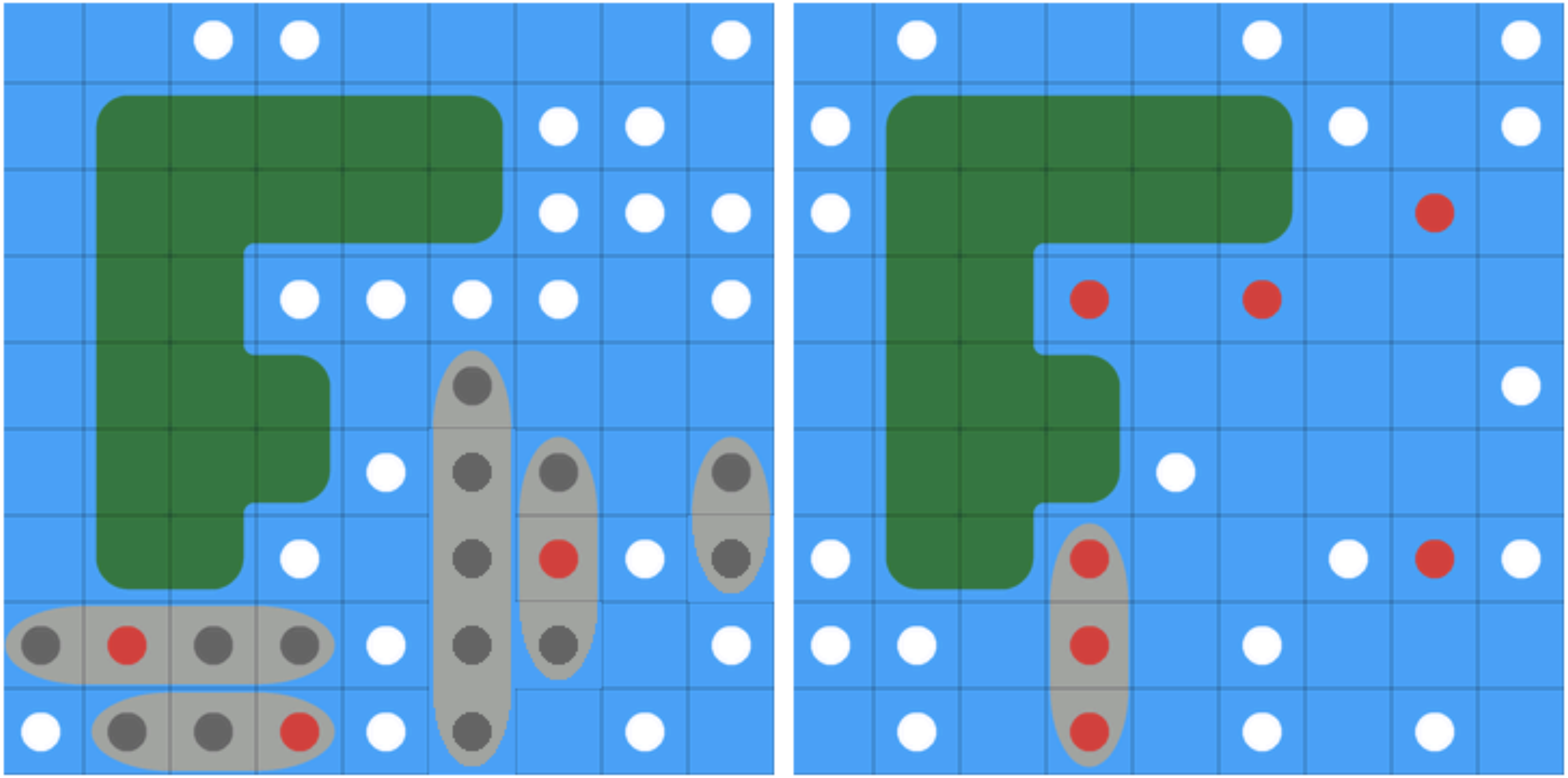
BFS algorithm

```python
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)
```

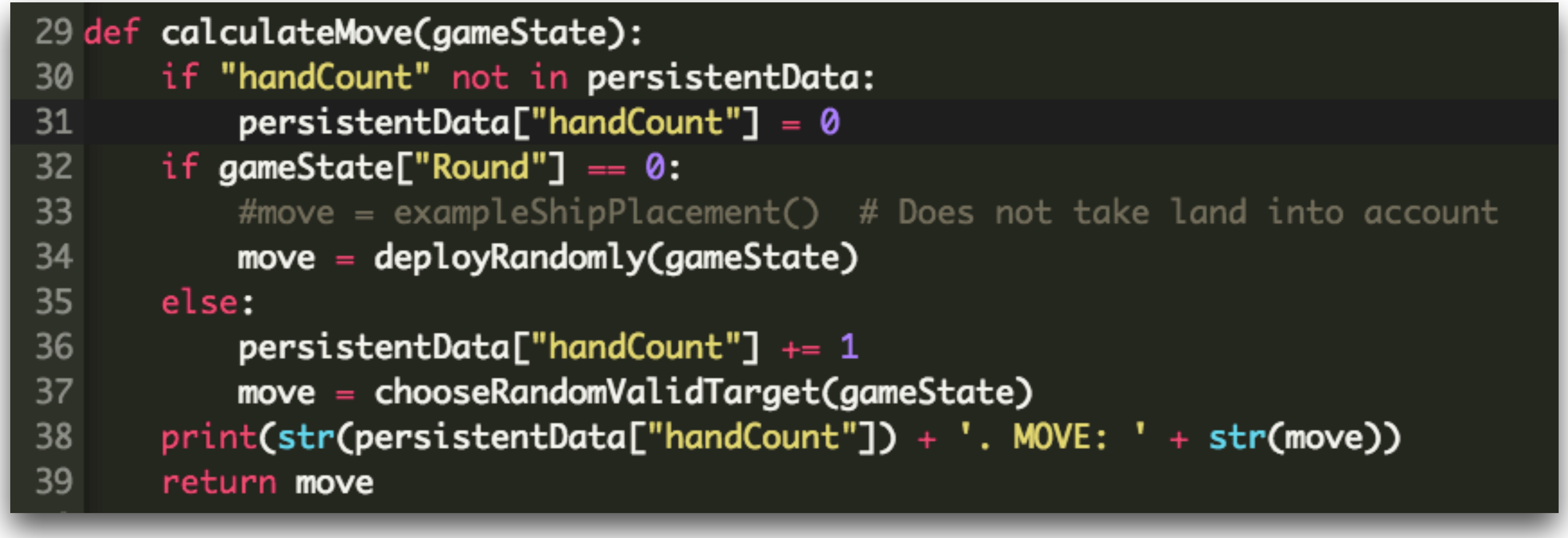
Search in Battleships

BATTLESHIPS

Battleships is an excellent, mid level introduction to developing game playing bots. More complicated than Noughts and Crosses, but less of a challenge than Texas Hold 'Em, Battleships strategy is more easily defined, and, ultimately, easier to implement.



Battleships



```python
def calculateMove(gameState):
    if "handCount" not in persistentData:
        persistentData["handCount"] = 0
    if gameState["Round"] == 0:
        #move = exampleShipPlacement()  # Does not take land into account
        move = deployRandomly(gameState)
    else:
        persistentData["handCount"] += 1
        move = chooseRandomValidTarget(gameState)
    print(str(persistentData["handCount"]) + '. MOVE: ' + str(move))
    return move
```

```
>>> print('Game state: ' + str(gameState))
```

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				
		●		X
X				

X is a missed shot,

● is a hit ship.

Search in Target mode

X				
		1		
	4	●	2	X
X		3		

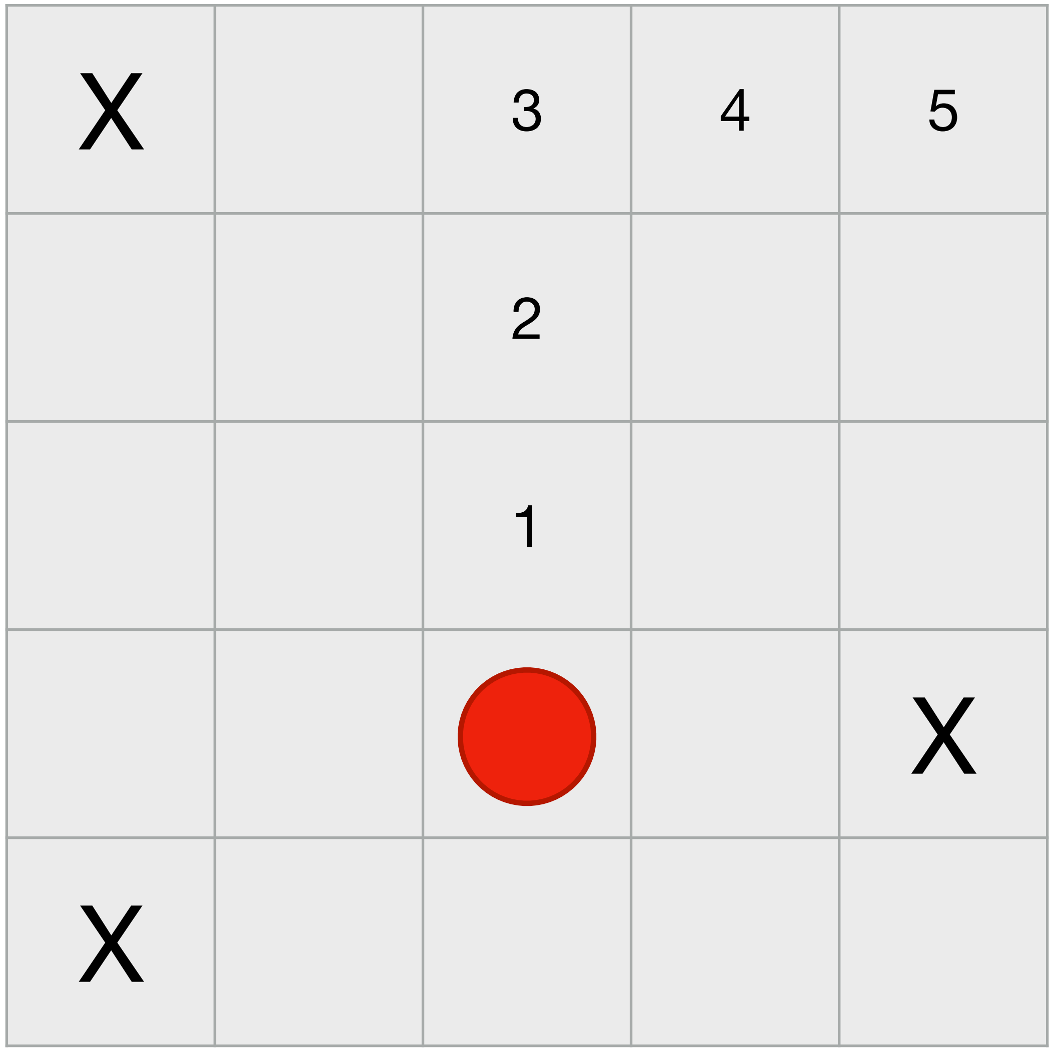
BFS will first consider all neighbours of the start node, even if the next hit is discovered.

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.

Search in Target mode



X		3	4	5
		2		
		1		
		●		X
X				

DFS, on the other hand, will follow a path from the start node, even if the next hit is not discovered.

Search in Target mode

X		3	4	5
		2		
		X		
		●		X
X				

DFS, on the other hand, will follow a path from the start node, even if the next hit is not discovered.

Search in Target mode

X		3	4	5
		X		
		X		
		●		X
X				

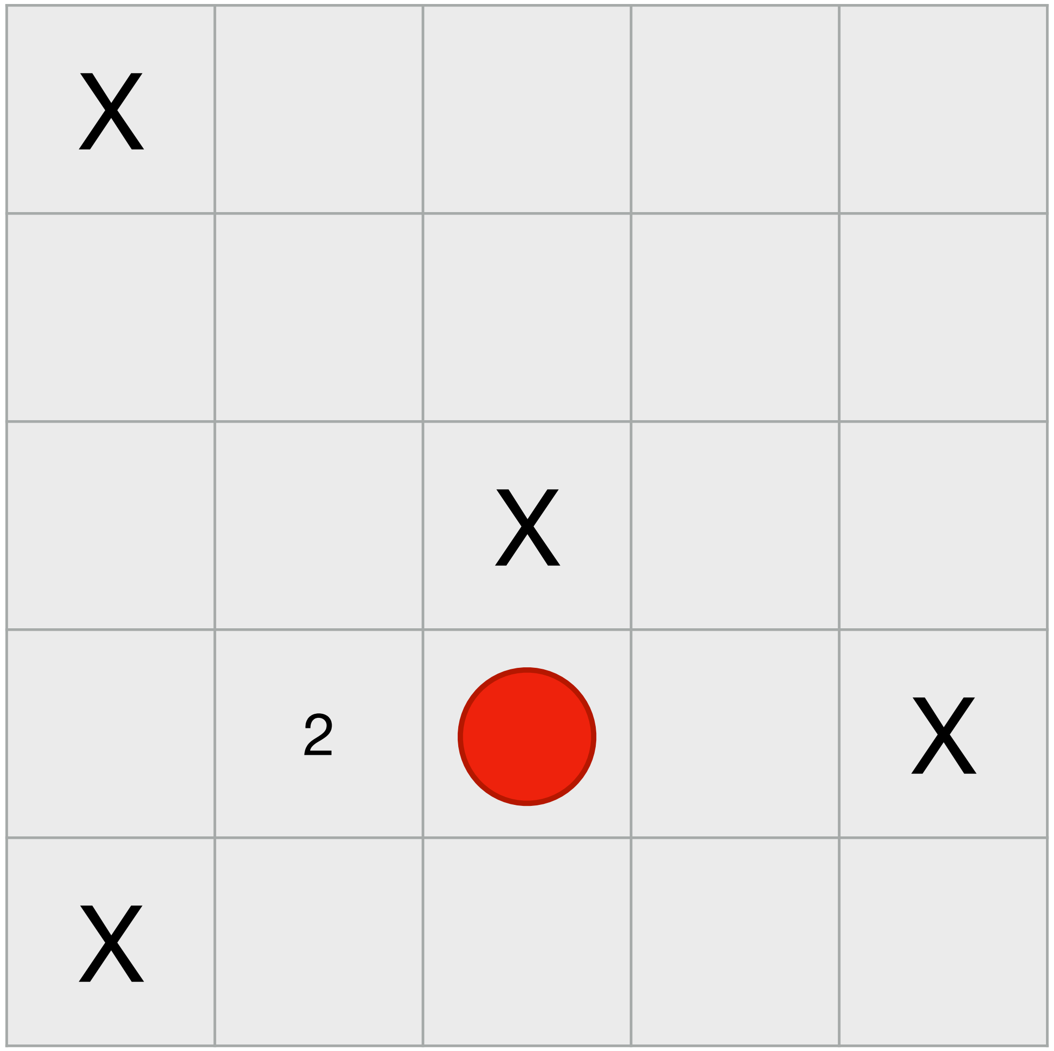
DFS, on the other hand, will follow a path from the start node, even if the next hit is not discovered.

Search in Target mode

X				
		1		
		●		X
X				

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.

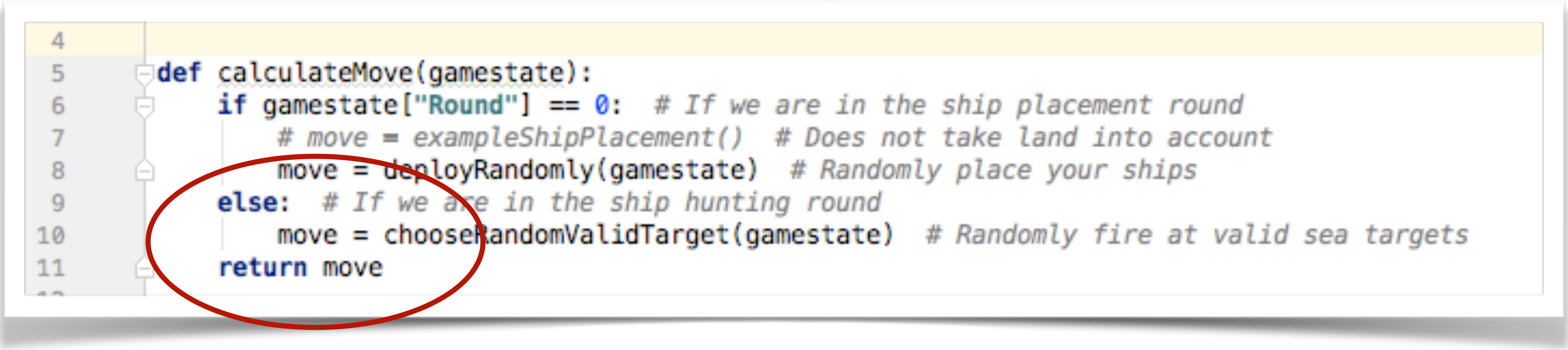
Search in Target mode

X				
		X		
3	●	●		X
X				

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

The important part

- function *calculateMove*



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

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

Coding: integrate DFS with pruning strategy into the battleships code

TODO list

1. memorise your previous move in the *persistentData*

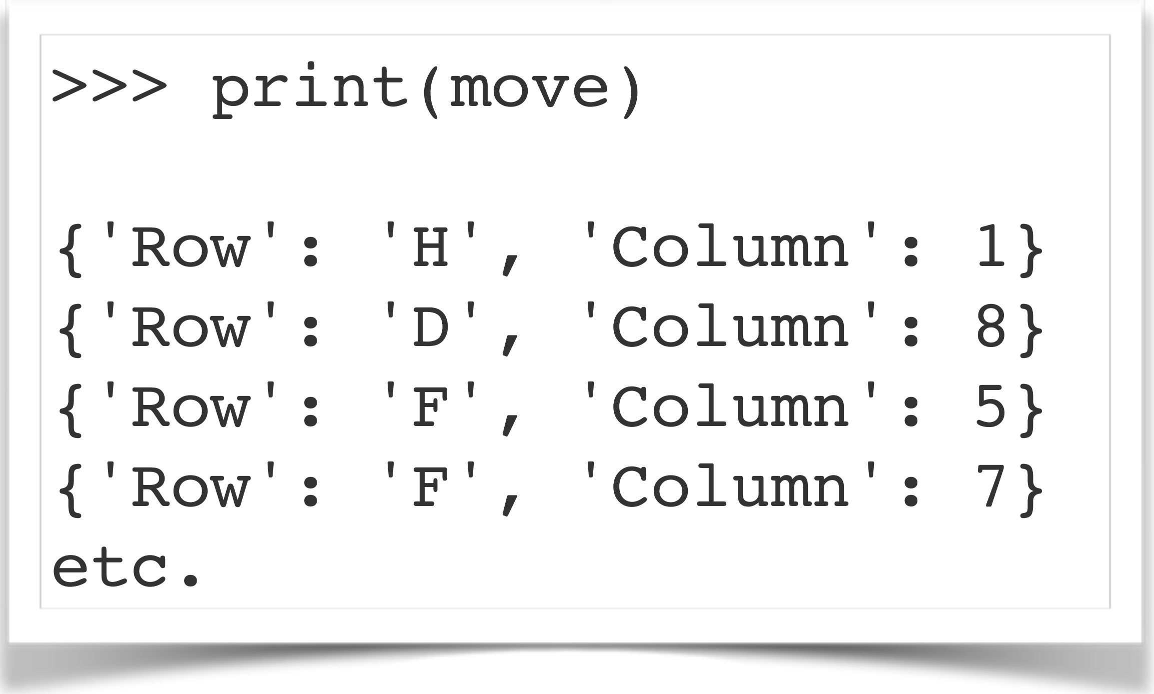
2. check whether the move from *persistentData* was a hit (using *gameState*)

3. add an *if-else* switch between the hunt mode and the target mode

4. for the target mode, implement DFS with pruning

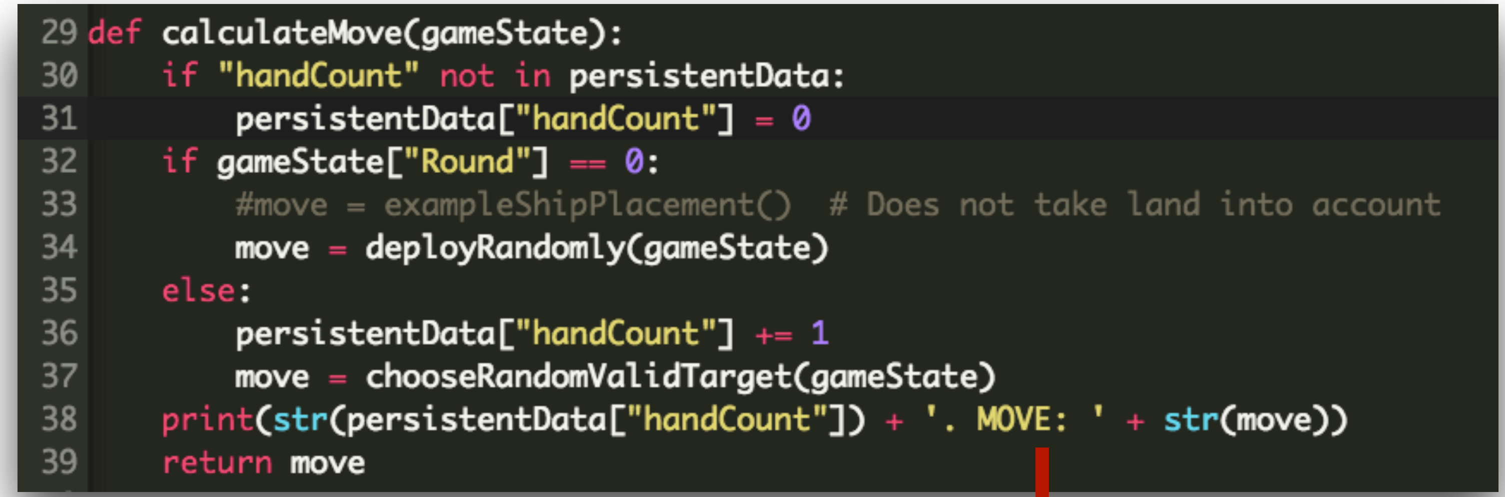
Step 1: memorising previous move

In the code, a move is represented by a dictionary:



```
>>> print(move)

{'Row': 'H', 'Column': 1}
{'Row': 'D', 'Column': 8}
{'Row': 'F', 'Column': 5}
{'Row': 'F', 'Column': 7}
etc.
```

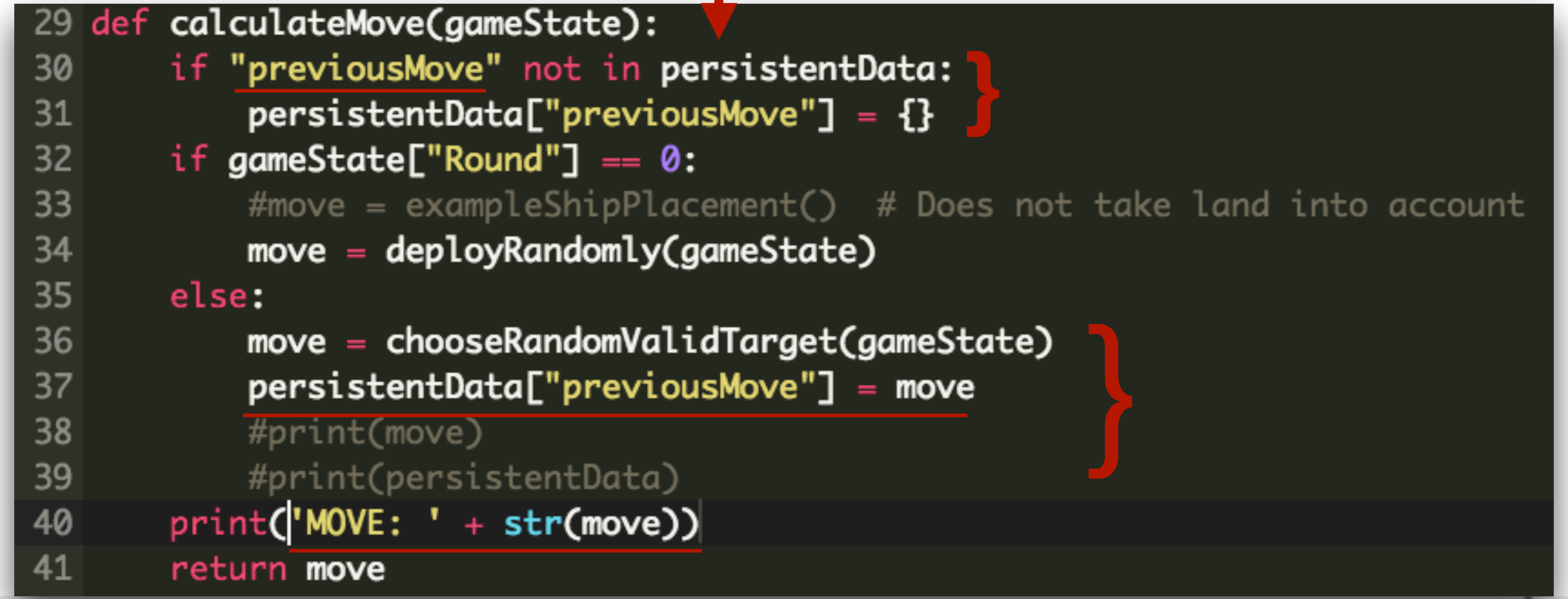
Step 1: memorising previous move



```python
def calculateMove(gameState):
    if "handCount" not in persistentData:
        persistentData["handCount"] = 0
    if gameState["Round"] == 0:
        #move = exampleShipPlacement()  # Does not take land into account
        move = deployRandomly(gameState)
    else:
        persistentData["handCount"] += 1
        move = chooseRandomValidTarget(gameState)
    print(str(persistentData["handCount"]) + '. MOVE: ' + str(move))
    return move
```

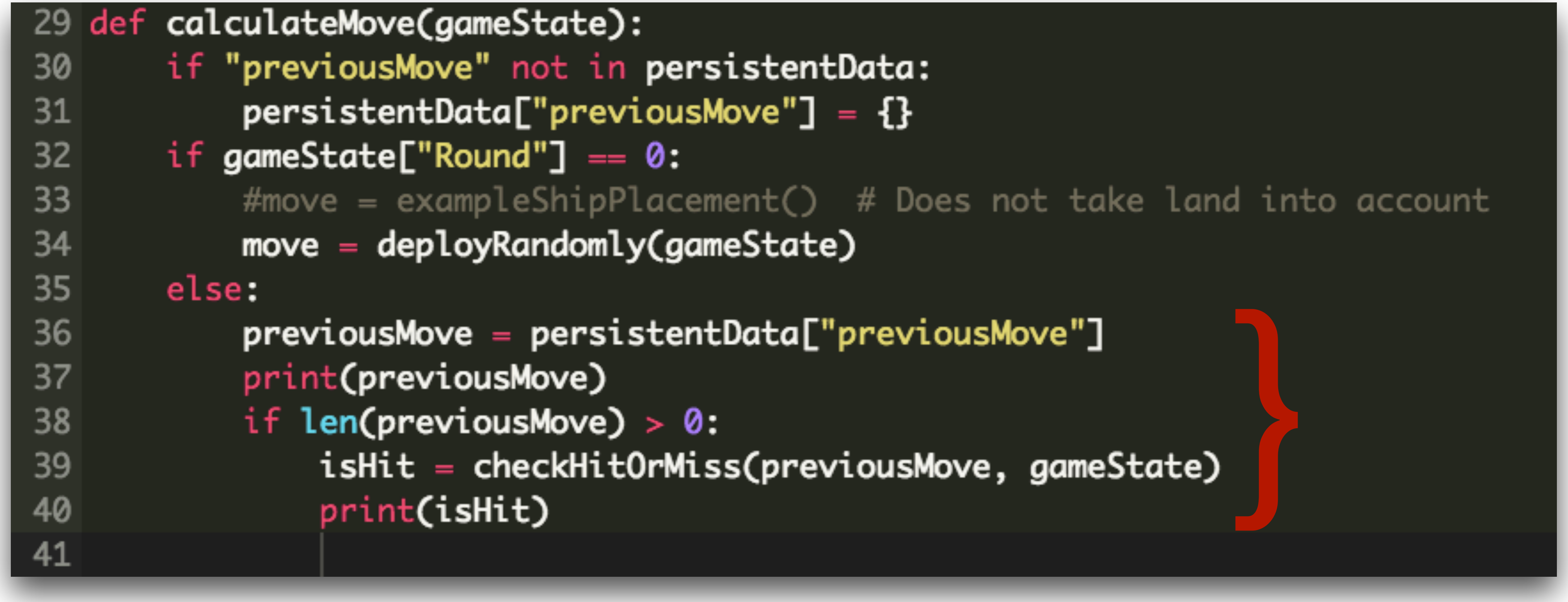
default code

updated code



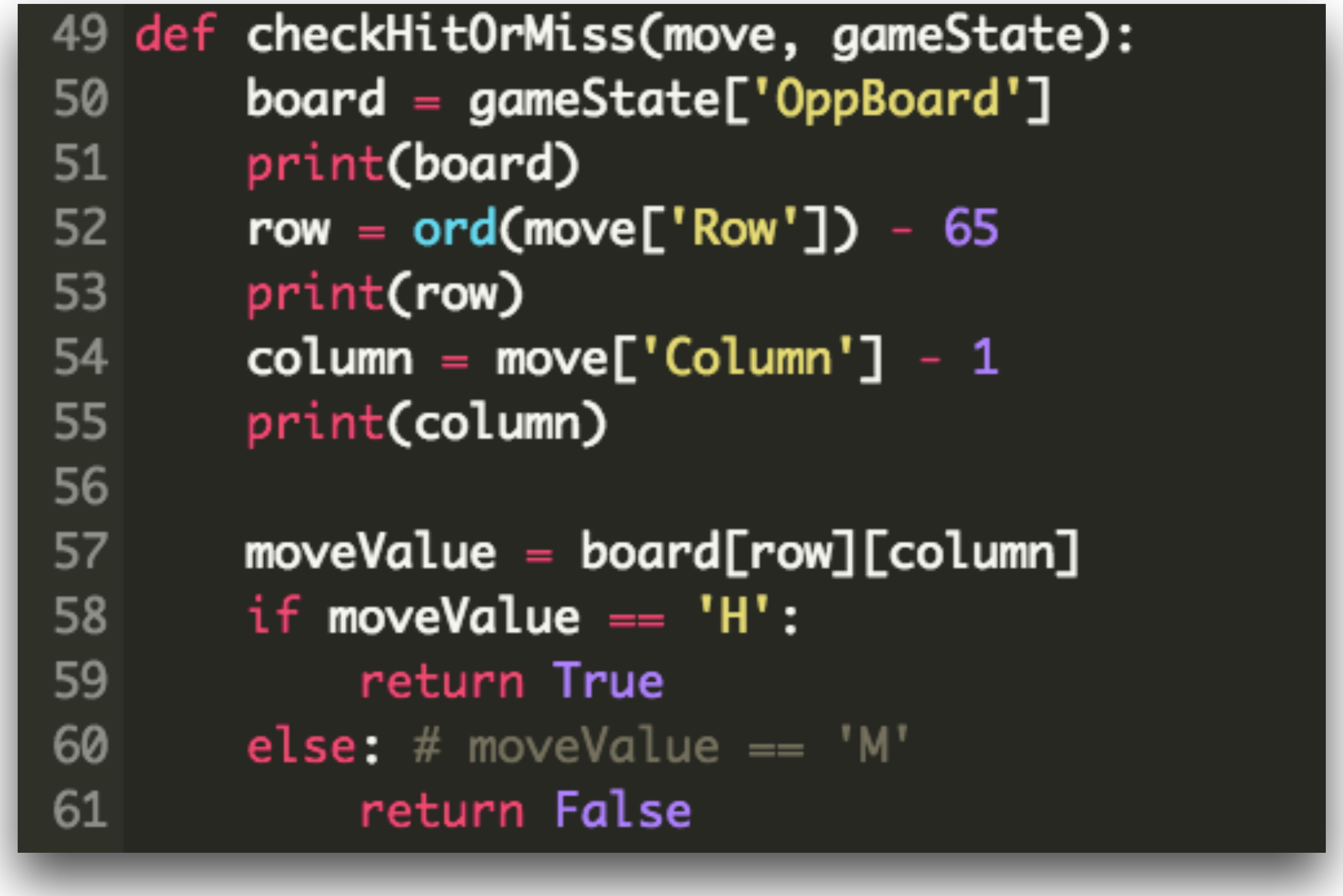
```python
def calculateMove(gameState):
    if "previousMove" not in persistentData:
        persistentData["previousMove"] = {}
    if gameState["Round"] == 0:
        #move = exampleShipPlacement()  # Does not take land into account
        move = deployRandomly(gameState)
    else:
        move = chooseRandomValidTarget(gameState)
        persistentData["previousMove"] = move
        #print(move)
        #print(persistentData)
    print('MOVE: ' + str(move))
    return move
```

Step 2: checking previous move

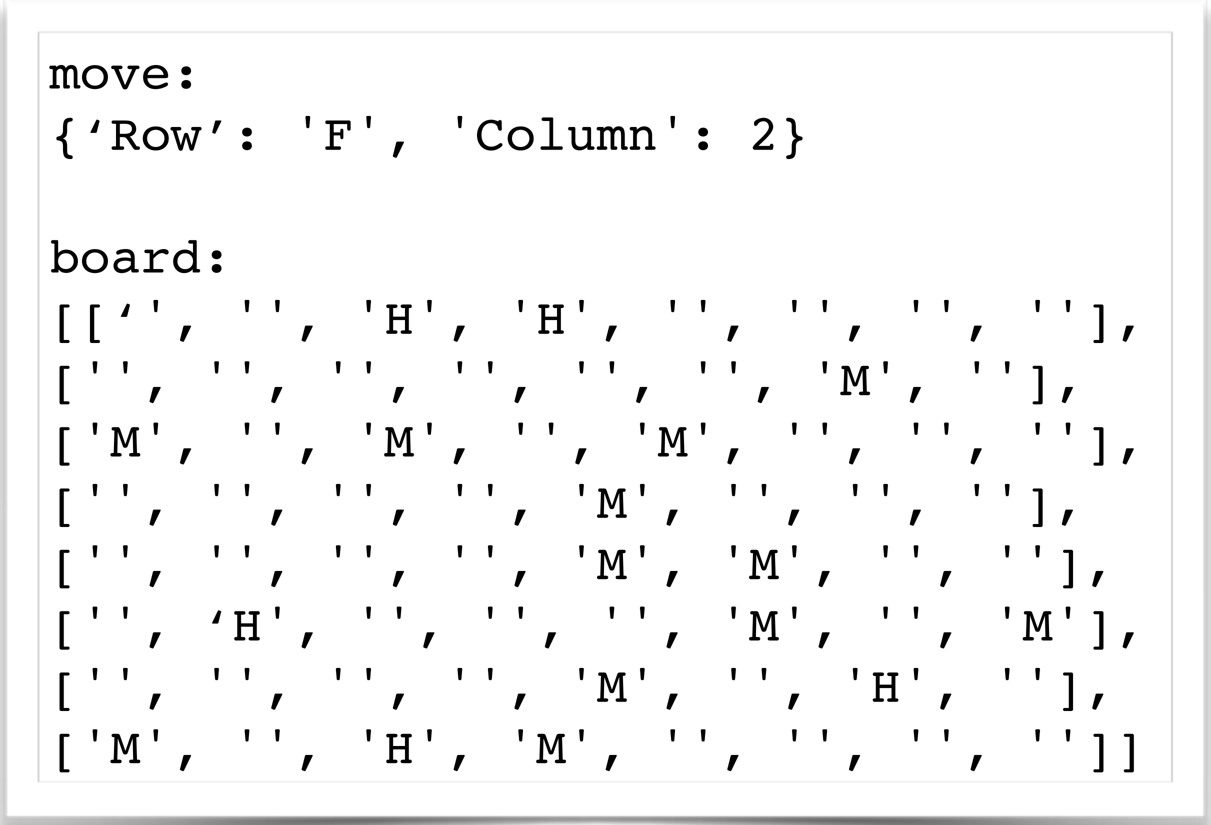


```python
def calculateMove(gameState):
    if "previousMove" not in persistentData:
        persistentData["previousMove"] = {}
    if gameState["Round"] == 0:
        #move = exampleShipPlacement()  # Does not take land into account
        move = deployRandomly(gameState)
    else:
        previousMove = persistentData["previousMove"]
        print(previousMove)
        if len(previousMove) > 0:
            isHit = checkHitOrMiss(previousMove, gameState)
            print(isHit)

```

Step 2: checking previous move

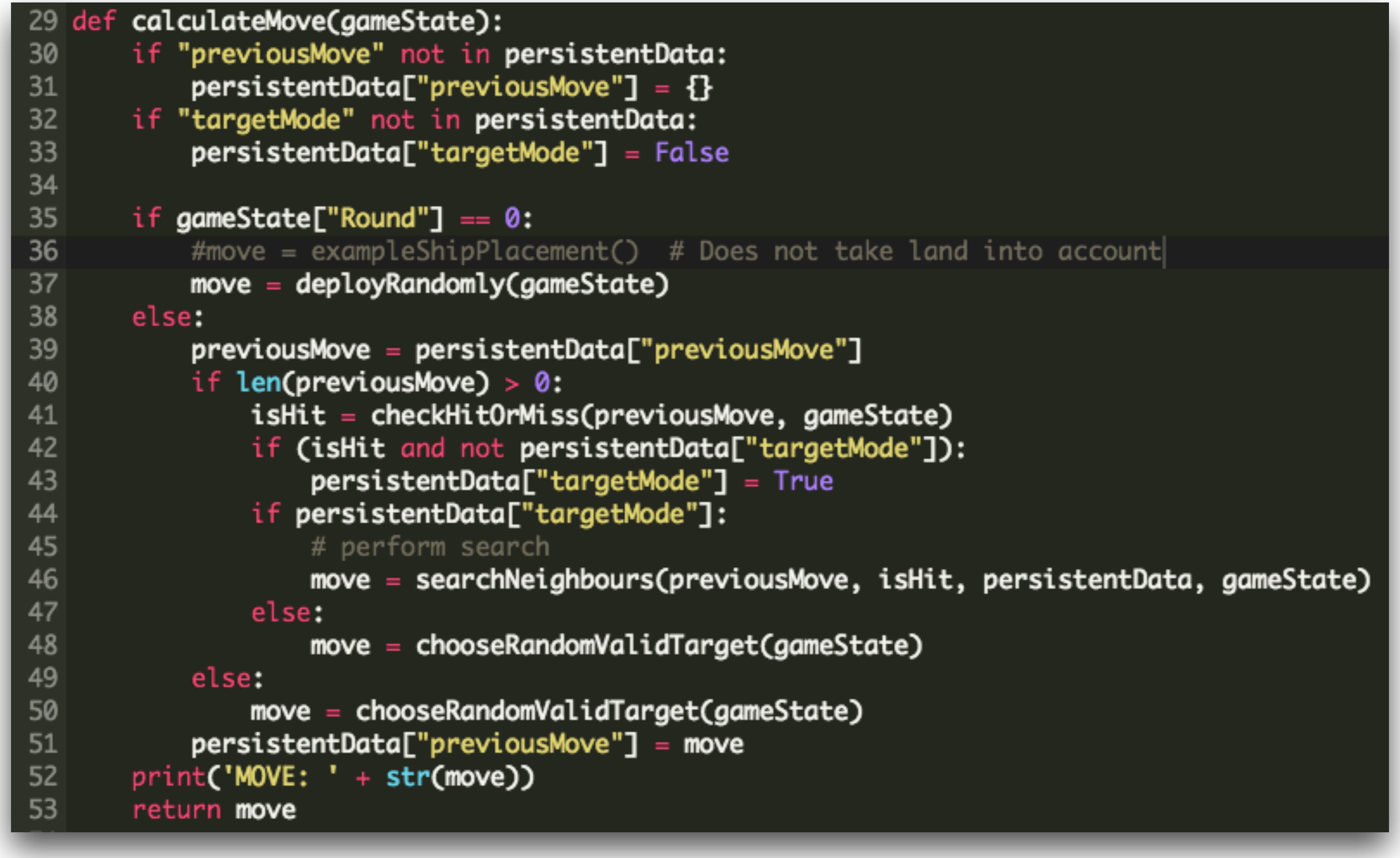


```python
def checkHitOrMiss(move, gameState):
    board = gameState['OppBoard']
    print(board)
    row = ord(move['Row']) - 65
    print(row)
    column = move['Column'] - 1
    print(column)

    moveValue = board[row][column]
    if moveValue == 'H':
        return True
    else: # moveValue == 'M'
        return False
```



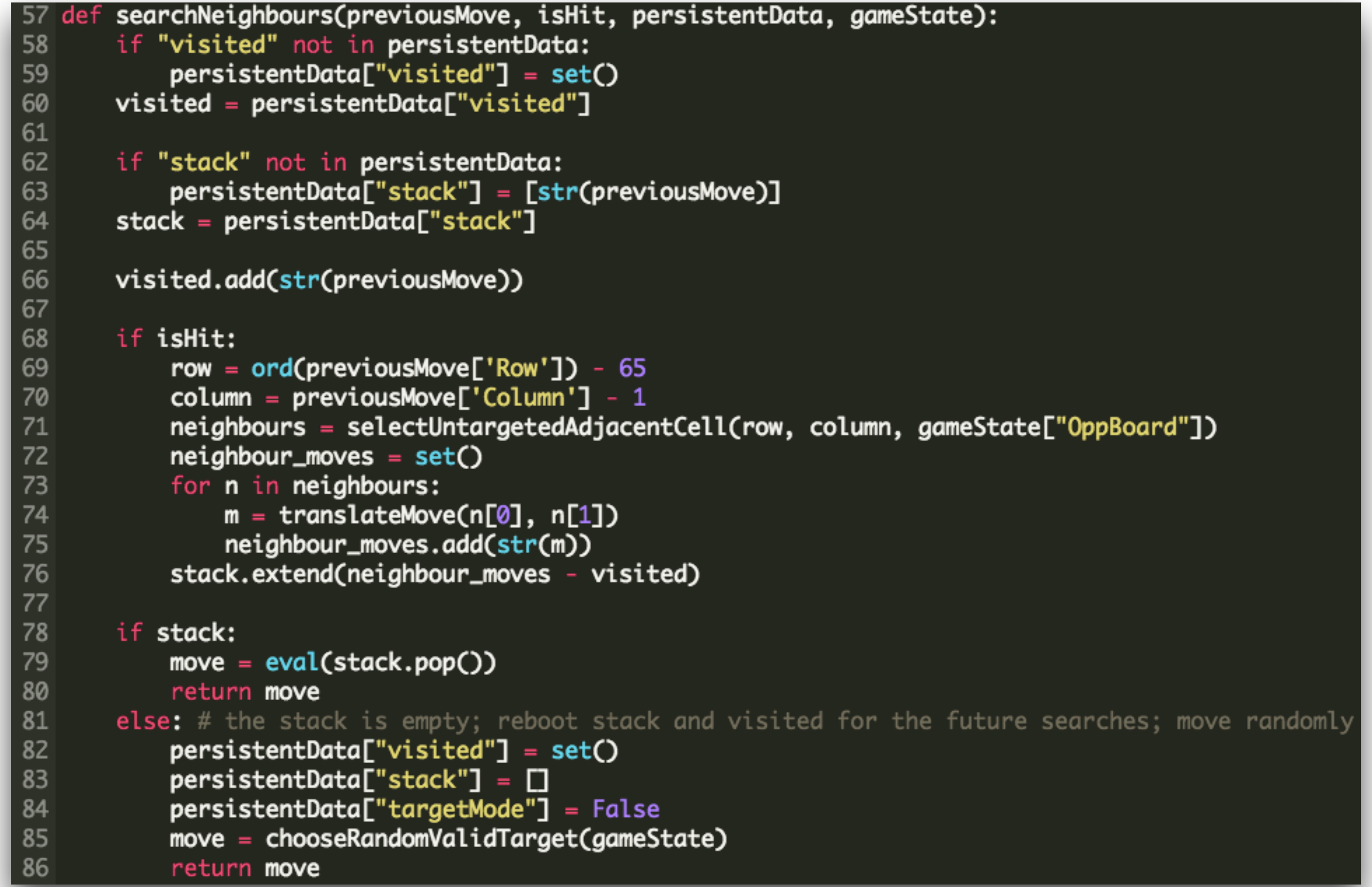
```
move:
{‘Row’: 'F', 'Column': 2}

board:
[[‘', '', 'H', 'H', '', '', '', ''],
['', '', '', '', '', '', 'M', ''],
['M', '', 'M', '', 'M', '', '', ''],
['', '', '', '', 'M', '', '', ''],
['', '', '', '', 'M', 'M', '', ''],
['', ‘H', '', '', '', 'M', '', 'M'],
['', '', '', '', 'M', '', 'H', ''],
['M', '', 'H', 'M', '', '', '', '']]
```

Step 3: putting the logic together



```python
def calculateMove(gameState):
    if "previousMove" not in persistentData:
        persistentData["previousMove"] = {}
    if "targetMode" not in persistentData:
        persistentData["targetMode"] = False

    if gameState["Round"] == 0:
        #move = exampleShipPlacement()  # Does not take land into account
        move = deployRandomly(gameState)
    else:
        previousMove = persistentData["previousMove"]
        if len(previousMove) > 0:
            isHit = checkHitOrMiss(previousMove, gameState)
            if (isHit and not persistentData["targetMode"]):
                persistentData["targetMode"] = True
            if persistentData["targetMode"]:
                # perform search
                move = searchNeighbours(previousMove, isHit, persistentData, gameState)
            else:
                move = chooseRandomValidTarget(gameState)
        else:
            move = chooseRandomValidTarget(gameState)
        persistentData["previousMove"] = move
    print('MOVE: ' + str(move))
    return move
```

Step 4: adding DFS with pruning



```python
def searchNeighbours(previousMove, isHit, persistentData, gameState):
    if "visited" not in persistentData:
        persistentData["visited"] = set()
    visited = persistentData["visited"]

    if "stack" not in persistentData:
        persistentData["stack"] = [str(previousMove)]
    stack = persistentData["stack"]

    visited.add(str(previousMove))

    if isHit:
        row = ord(previousMove['Row']) - 65
        column = previousMove['Column'] - 1
        neighbours = selectUntargetedAdjacentCell(row, column, gameState["OppBoard"])
        neighbour_moves = set()
        for n in neighbours:
            m = translateMove(n[0], n[1])
            neighbour_moves.add(str(m))
        stack.extend(neighbour_moves - visited)

    if stack:
        move = eval(stack.pop())
        return move
    else: # the stack is empty; reboot stack and visited for the future searches; move randomly
        persistentData["visited"] = set()
        persistentData["stack"] = []
        persistentData["targetMode"] = False
        move = chooseRandomValidTarget(gameState)
        return move
```