CSE4082 AI PROJECT 1

Samet Enes Örsdemir

150119661

This is “State” class to represent nodes.

```
class State:
    def __init__(self, holes, map, depth, parent):
        self.holes = holes
        self.map = map
        self.depth = depth
        self.parent = parent

    def print_board(self):
        i = 0
        print("Depth:" + str(self.depth))
        for row in self.map:
            print("")
            for column in row:
                if(column == 1):
                    print("X", end = ' ')
                elif(column == 0):
                    print("O", end = ' ')
                else:
                    print(" ", end = ' ')
            i += 1
```

```
ime_limit = 0
frontier_list=[]

initial_state = State([[3,3]],
           [
            [8,8,1,1,1,8,8],
            [8,8,1,1,1,8,8],
            [1,1,1,1,1,1,1],
            [1,1,1,0,1,1,1],
            [1,1,1,1,1,1,1],
            [8,8,1,1,1,8,8],
            [8,8,1,1,1,8,8]
            ], 0, None)
```

```
goal_state_map =  [
            [8,8,0,0,0,8,8],
            [8,8,0,0,0,8,8],
            [0,0,0,0,0,0,0],
            [0,0,0,1,0,0,0],
            [0,0,0,0,0,0,0],
            [8,8,0,0,0,8,8],
            [8,8,0,0,0,8,8]
            ]

frontier_list.append(initial_state)
```

The search algorithms were implemented using only the frontier list. No other structure was used.

Expand function for BFS, DFS and Iterative Deepening:

```
def expand_state(state,frontier): # expand for dfs bfs iterative
    for hole in state.holes: # holes in state
        x = hole[0] # x coordinate of hole
        y = hole[1] # y coordinate of hole
        parent = state # parent assign
        depth = int(state.depth + 1) # depth assign
        if((y!=0) and (y!=1)): # map limit
            if ((state.map[y-1][x]== 1) and (state.map[y-2][x]==1)): #
north check
                map = copy.deepcopy(state.map) # map copy
                map[y][x] = 1 # hole update
                map[y-1][x] = 0 # hole update
                map[y-2][x] = 0 # hole update
                new_holes = [[x,y-1],[x,y-2]] # holes list update
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                frontier.append(State( holes, map, depth, parent)) #
add child to frontier
        if((x != 0) and (x != 1)): # map limit
            if ((state.map[y][x-1]== 1) and (state.map[y][x-2]==1)): #
west check
                map2 = copy.deepcopy(state.map)
```

```
                map2[y][x] = 1
                map2[y][x-1] = 0
                map2[y][x-2] = 0
                new_holes = [[x-1,y],[x-2,y]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                frontier.append(State( holes, map2, depth, parent))  #
add child to frontier
        if((x != 5) and (x != 6)): # map limit
            if ((state.map[y][x+1]== 1) and (state.map[y][x+2]==1)): #
east check
                map2 = copy.deepcopy(state.map)
                map2[y][x] = 1
                map2[y][x+1] = 0
                map2[y][x+2] = 0
                new_holes = [[x+1,y],[x+2,y]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                frontier.append(State( holes, map2, depth, parent)) #
add child to frontier
        if((y!=5) and (y!=6)): # map limit
            if ((state.map[y+1][x]== 1) and (state.map[y+2][x]==1)): #
south check
                map2 = copy.deepcopy(state.map)
                map2[y][x] = 1
                map2[y+1][x] = 0
                map2[y+2][x] = 0
                new_holes = [[x,y+1],[x,y+2]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                frontier.append(State( holes, map2, depth, parent)) #
add child to frontier
```

BFS Implementation:

```
def bfs_search(frontier): #bfs
    current_depth = 0 # current search depth
    expanded_counter = 0 # counter for expanded nodes
    frontier_max = 0 # max frontier length
    start = time.time() # starting time
    best_solution_in_our_hand = initial_state # best solution for t=0
    while (True): # loop
        print("\n")
        print(len(frontier))
        # if condition for update max frontier length
        if(len(frontier) > frontier_max): frontier_max = len(frontier)
        time_processed = time.time() - start # time update
        if time_processed >= time_limit: # time limit check
            return best_solution_in_our_hand, time_processed,
expanded_counter,frontier_max
        state = frontier.pop(0) # frontier pop
        if(state.map == goal_state_map): # if con. for goal state check
            print("Successful!!!!")
            return state, time_processed, expanded_counter,
frontier_max
        else:
            if(state.depth == current_depth): # if depth is same
                state.print_board()      # print board
            else:
                best_solution_in_our_hand = state #update best solution
                print("---->")
                state.print_board()
                current_depth += 1
            expand_state(state,frontier) # expand state
            expanded_counter += 1
```

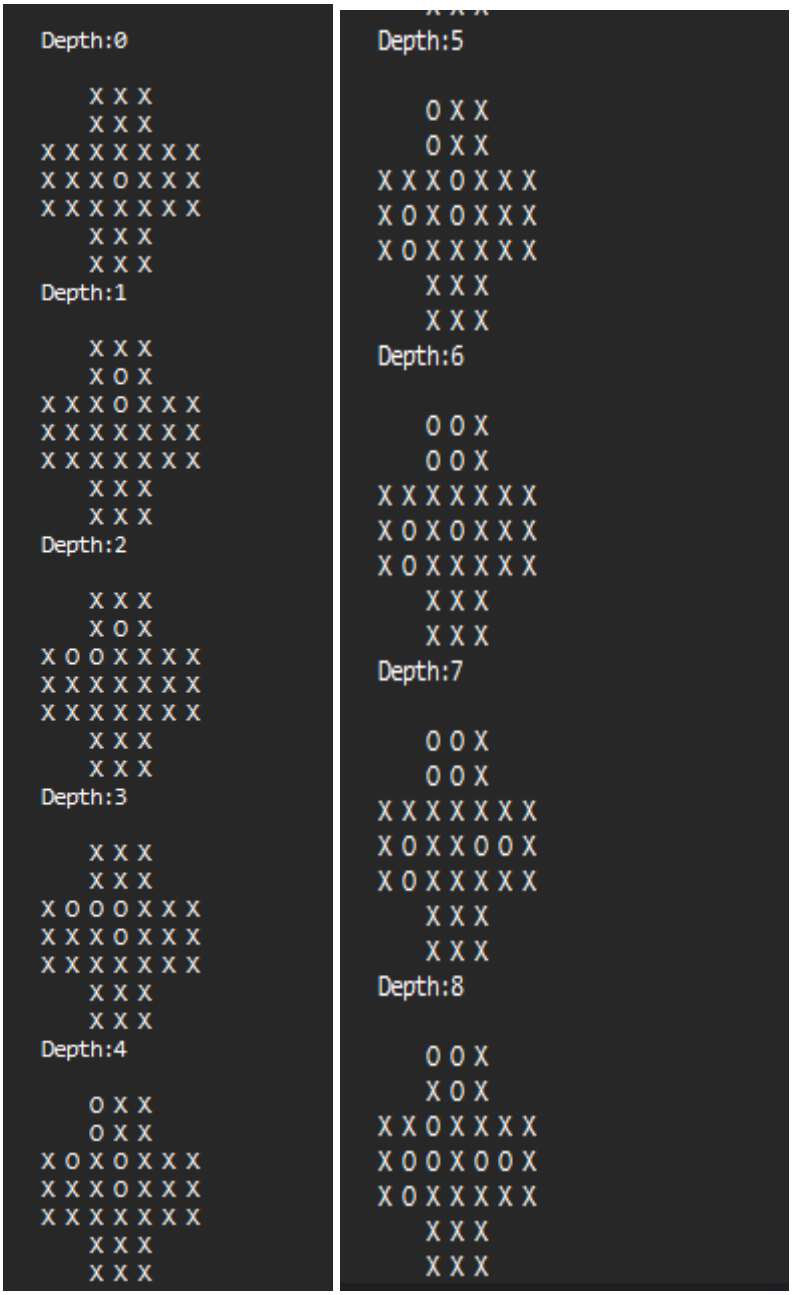
BFS Test:

The BFS algorithm was run for 60 minutes.

```
<<<PEG SOLITAIRE SOLVER AI>>>
-----------------------------
a. Breadth-First Search
b. Depth-First Search
c. Iterative Deepening Search
d. Depth-First Search with Random Selection
e. Depth-First Search with a Node Selection Heur
istic


Select a method: a
Type your time limit (in minutes): 60
```

Result:

```
Depth:8

    O O X
    X O X
X X O X X X X
X O O X O O X
X O X X X X X
    X X X
    X X X

Sub-optimum Solution Found with 24 remaining pegs
Time spent: 60 minutes
465206 nodes expanded during the search.
Maximum 3989764 nodes stored in the memory during the search
.
```



In some runs, it may give an Out of Memory error.

DFS Implementation:

Same method as BFS. The difference is that the frontier list was popped from the end.

```
def dfs_search(frontier): # dfs
    current_depth = 0 # current search depth
    expanded_counter = 0 # counter for expanded nodes
    frontier_max = 0 # max frontier length
    start = time.time() # starting time
    best_solution_in_our_hand = initial_state # best solution for t=0
    while (True):
        print("\n")
        print(len(frontier))
        if(len(frontier) > frontier_max): frontier_max = len(frontier)
        time_processed = time.time() - start
        if time_processed >= time_limit:
            return best_solution_in_our_hand, time_processed,
expanded_counter, frontier_max
        state = frontier.pop(len(frontier)-1) # frontier pop from end
of the list
        if(state.map == goal_state_map): # goal state check
            print("Successful!!!!")
            state.print_board()
            return state, time_processed, expanded_counter,
frontier_max
        else:
            if(state.depth == current_depth):
                state.print_board()
                pass
            else:
                print("---->")
                if(state.depth > best_solution_in_our_hand.depth): #
best solution update
                    best_solution_in_our_hand = state
                state.print_board()
                current_depth += 1
            expand_state(state,frontier) # expand state function
            expanded_counter += 1
```

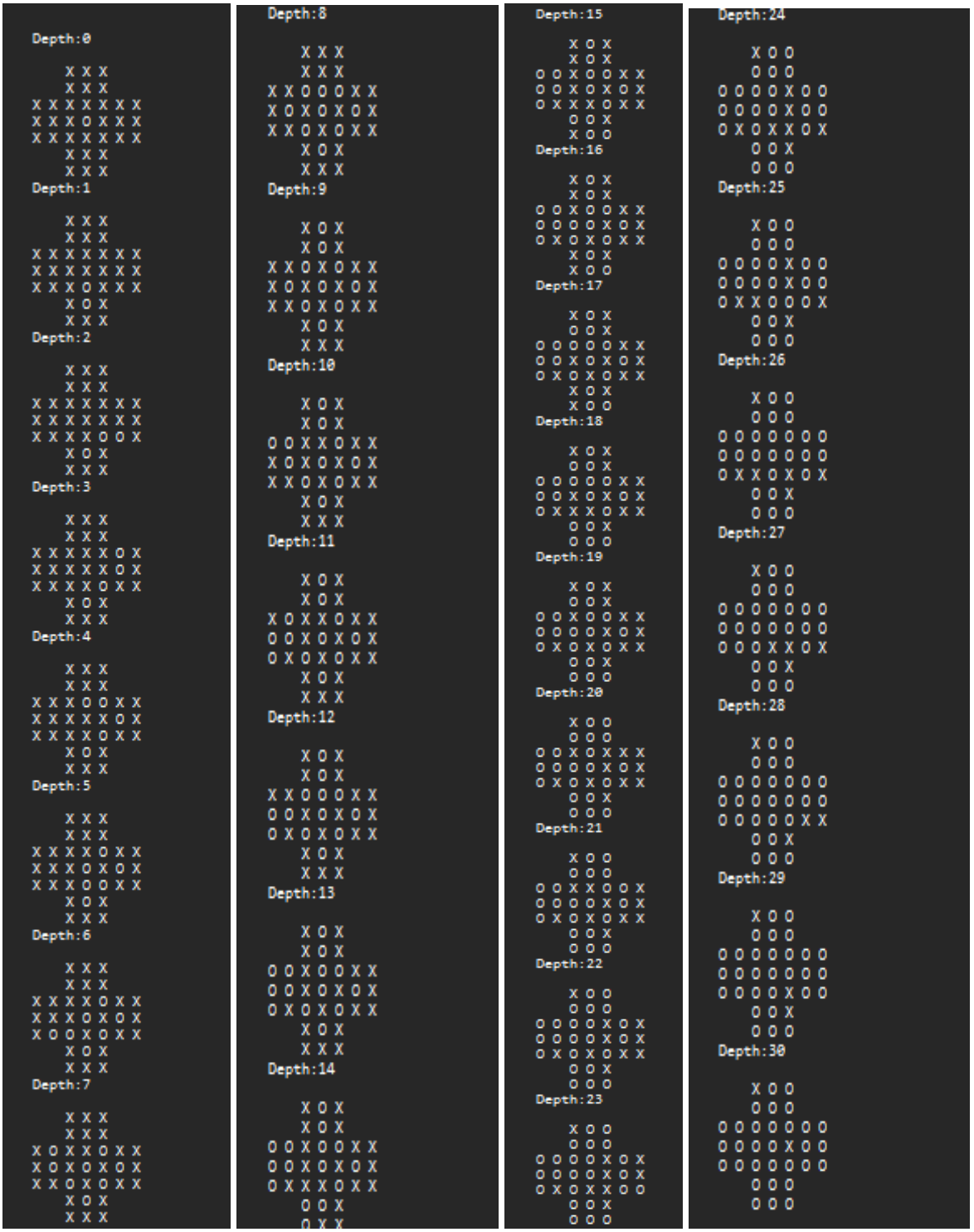
DFS Test:

```
<<<PEG SOLITAIRE SOLVER AI>>>
----------------------------
a. Breadth-First Search
b. Depth-First Search
c. Iterative Deepening Search
d. Depth-First Search with Random Selection
e. Depth-First Search with a Node Selection Heuristic


Select a method: b
Type your time limit (in minutes): 60
```

The DFS algorithm was run for 60 minutes.

Result:

```
Depth:30

    X O O
    O O O
O O O O O O O
O O O O X O O
O O O O O O O
    O O O
    O O O

Sub-optimum Solution Found with 2 remaining pegs
Time spent: 60 minutes
45203106 nodes expanded during the search.
Maximum 161 nodes stored in the memory during the search.
```


Depth:0
Depth:1
Depth:2
Depth:3
Depth:4
Depth:5
Depth:6
Depth:7
Depth:8
Depth:9
Depth:10
Depth:11
Depth:12
Depth:13
Depth:14
Depth:15
Depth:16
Depth:17
Depth:18
Depth:19
Depth:20
Depth:21
Depth:22
Depth:23
Depth:24
Depth:25
Depth:26
Depth:27
Depth:28
Depth:29
Depth:30

Iterative Deepening Search Implementaton:

```
def iterative_deepening_search(frontier): # iterative deepening
    current_depth = 0 # current search depth
    expanded_counter = 0 # counter for expanded nodes
    frontier_max = 0 # max frontier length
    start = time.time() # starting time
    best_solution_in_our_hand = initial_state # best solution for t=0
    depth_limit = 0 # depth limit for iteration
    while (True):
        print("\n")
        print("frontier length: " + str(len(frontier)))
        if(len(frontier) > frontier_max): frontier_max = len(frontier)
        time_processed = time.time() - start
        if time_processed >= time_limit: # time check
            return best_solution_in_our_hand, time_processed,
expanded_counter, frontier_max

        state = frontier.pop(len(frontier)-1) # pop from end of the
list
        if(state.map == goal_state_map): # goal state check
            print("Successful!!!!")
            state.print_board()
            return state, time_processed, expanded_counter,frontier_max
        else:
            state.print_board()
            if(state.depth == depth_limit): # depth limit check
                if(state.depth > best_solution_in_our_hand.depth):
                    best_solution_in_our_hand = state
                if(len(frontier)!=0): # is frontier empty or not?
                    continue
                else:
                    print("\n\n*********iterative restart*******\n\n")
                    print("*********depth limit: " + str(depth_limit) +
"**********\n")
                    depth_limit += 1 # depth limit increment
                    frontier.clear() # frontier clear to restart
                    frontier.append(initial_state)
                    continue
            current_depth += 1
            expand_state(state,frontier) # expand state
            expanded_counter += 1
```

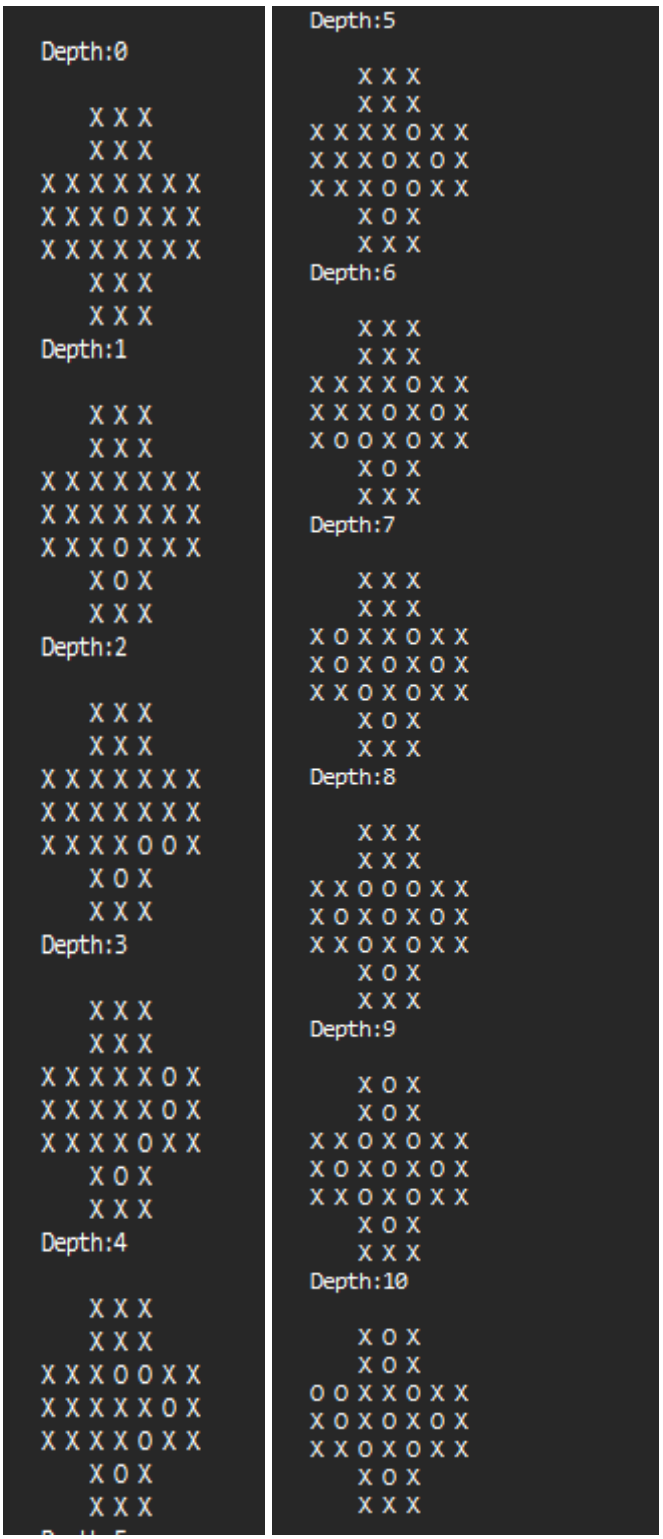
Iterative Deepening Search Test:

The Iterative Deepening Search algorithm was run for 60 minutes.

```
<<<PEG SOLITAIRE SOLVER AI>>>
-----------------------------
a. Breadth-First Search
b. Depth-First Search
c. Iterative Deepening Search
d. Depth-First Search with Random Selection
e. Depth-First Search with a Node Selection Heuristic


Select a method: c
Type your time limit (in minutes): 60
```

Result:

```
Depth:10

    X O X
    X O X
O O X X O X X
X O X O X O X
X X O X O X X
    X O X
    X X X

Sub-optimum Solution Found with 22 remaining pegs
Time spent: 60 minutes
8926976 nodes expanded during the search.
Maximum 98 nodes stored in the memory during the search.
```


Depth:0
X X X
X X X
X X X X X X X
X X X O X X X
X X X X X X X
X X X
X X X
Depth:1
X X X
X X X
X X X X X X X
X X X X X X X
X X X O X X X
X O X
X X X
Depth:2
X X X
X X X
X X X X X X X
X X X X X X X
X X X X O O X
X O X
X X X
Depth:3
X X X
X X X
X X X X X O X
X X X X X O X
X X X X O X X
X O X
X X X
Depth:4
X X X
X X X
X X X O O X X
X X X X X O X
X X X X O X X
X O X
X X X
Depth:5
X X X
X X X
X X X X O X X
X X X O X O X
X X X O O X X
X O X
X X X
Depth:6
X X X
X X X
X X X X O X X
X X X O X O X
X O O X O X X
X O X
X X X
Depth:7
X X X
X X X
X O X X O X X
X O X O X O X
X X O X O X X
X O X
X X X
Depth:8
X X X
X X X
X X O O O X X
X O X O X O X
X X O X O X X
X O X
X X X
Depth:9
X O X
X O X
X X O X O X X
X O X O X O X
X X O X O X X
X O X
X X X
Depth:10
X O X
X O X
O O X X O X X
X O X O X O X
X X O X O X X
X O X
X X X

DFS with Random Selection Implementation:

The difference compared to other algorithms is that the expand algorithm is different. Children created in the Expand function are placed at the end of the frontier list in random order.

```
def random_dfs_search(frontier):
    current_depth = 0 # current search depth
    expanded_counter = 0 # counter for expanded nodes
    frontier_max = 0 # max frontier length
    start = time.time() # starting time
    best_solution_in_our_hand = initial_state # best solution for t=0
    while (True):
        print("\n")
        time_processed = time.time() - start
        #print(time_processed)
        if(len(frontier) > frontier_max): frontier_max = len(frontier)
        if time_processed >= time_limit:
            return best_solution_in_our_hand, time_processed,
expanded_counter,frontier_max
        #print("\n")
        #print(len(frontier))
        state = frontier.pop(len(frontier)-1)
        if(state.map == goal_state_map):
            print("Successful!!!!")
            state.print_board()
            return state, time_processed, expanded_counter,
frontier_max
        else:
            if(state.depth == current_depth):
                state.print_board()
            else:
                print("---->")
                state.print_board()
                if(state.depth > best_solution_in_our_hand.depth):
                    best_solution_in_our_hand = state
                current_depth += 1
            expand_state_RANDOM(state,frontier)
            expanded_counter += 1
```

```
def expand_state_RANDOM(state,frontier):
    to_be_added = [] # children's temp list
    for hole in state.holes: # state holes
        x = hole[0] # hole coordinate for x
        y = hole[1] # hole coordinate for y
        parent = state # parent assign
        depth = int(state.depth + 1) # depth assign
        if((y!=0) and (y!=1)): # limit of map
            if ((state.map[y-1][x]== 1) and (state.map[y-2][x]==1)): #
north check
                map = copy.deepcopy(state.map)
                map[y][x] = 1 # map update
                map[y-1][x] = 0 # map update
                map[y-2][x] = 0 # map update
                new_holes = [[x,y-1],[x,y-2]] # holes update
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                to_be_added.append(State( holes, map, depth, parent)) #
add child to temp list
        if((y!=5) and (y!=6)): # limit of map
            if ((state.map[y+1][x]== 1) and (state.map[y+2][x]==1)): #
south check
                map2 = copy.deepcopy(state.map)
                map2[y][x] = 1
                map2[y+1][x] = 0
                map2[y+2][x] = 0
                new_holes = [[x,y+1],[x,y+2]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                to_be_added.append(State( holes, map2, depth, parent))
# add child to temp list
        if((x != 5) and (x != 6)): # limit of map
            if ((state.map[y][x+1]== 1) and (state.map[y][x+2]==1)): #
east check
                map2 = copy.deepcopy(state.map)
                map2[y][x] = 1
                map2[y][x+1] = 0
```

```
                map2[y][x+2] = 0
                new_holes = [[x+1,y],[x+2,y]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                to_be_added.append(State( holes, map2, depth, parent))
# add child to temp list
        if((x != 0) and (x != 1)): # limit of map
            if ((state.map[y][x-1]== 1) and (state.map[y][x-2]==1)): #
west check
                map2 = copy.deepcopy(state.map)
                map2[y][x] = 1
                map2[y][x-1] = 0
                map2[y][x-2] = 0
                new_holes = [[x-1,y],[x-2,y]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                to_be_added.append(State( holes, map2, depth, parent))
# add child to temp list
    while len(to_be_added) != 0:
        add_to_frontier = to_be_added.pop(random.randrange(0,
len(to_be_added))) # randomly add to frontier list
        frontier.append(add_to_frontier)
```

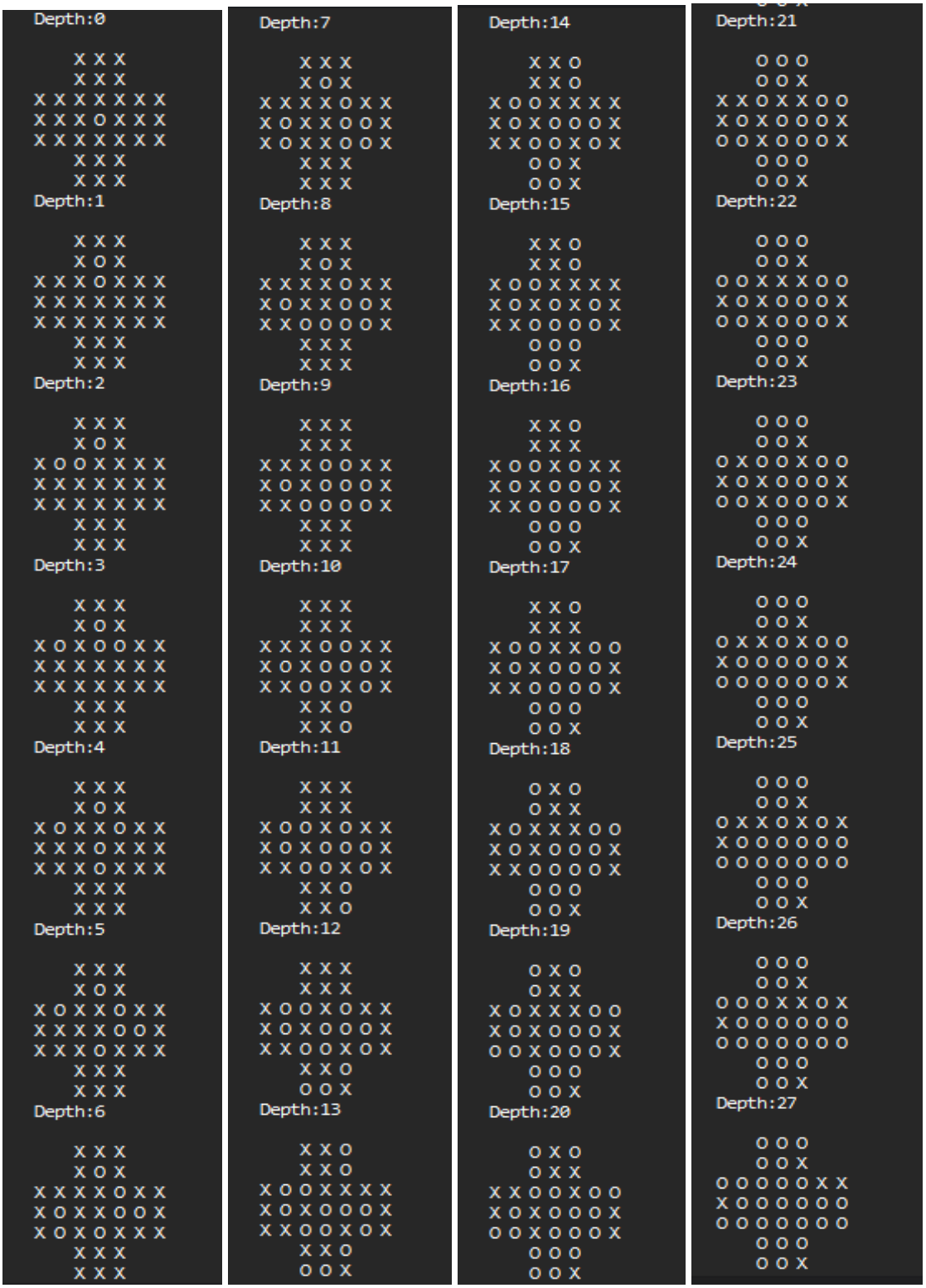
DFS with Random Selection Test:

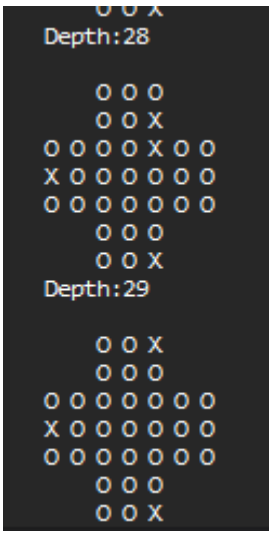
This algorithm was run for 60 minutes.

```
<<<PEG SOLITAIRE SOLVER AI>>>
-----------------------------
a. Breadth-First Search
b. Depth-First Search
c. Iterative Deepening Search
d. Depth-First Search with Random Selection
e. Depth-First Search with a Node Selection Heuristic


Select a method: d
Type your time limit (in minutes): 60
```

Result:

```
Sub-optimum Solution Found with 3 remaining pegs
Time spent: 60 minutes
45478616 nodes expanded during the search.
Maximum 173 nodes stored in the memory during the search.
```


Depth:0
X X X
X X X
X X X X X X X
X X X O X X X
X X X X X X X
X X X
X X X
Depth:1
X X X
X O X
X X X O X X X
X X X X X X X
X X X X X X X
X X X
X X X
Depth:2
X X X
X O X
X O O X X X X
X X X X X X X
X X X X X X X
X X X
X X X
Depth:3
X X X
X O X
X O X O O X X
X X X X X X X
X X X X X X X
X X X
X X X
Depth:4
X X X
X O X
X O X X O X X
X X X O X X X
X X X O X X X
X X X
X X X
Depth:5
X X X
X O X
X O X X O X X
X X X X O O X
X X X O X X X
X X X
X X X
Depth:6
X X X
X O X
X X X X O X X
X O X X O O X
X O X O X X X
X X X
X X X
Depth:7
X X X
X O X
X X X X O X X
X O X X O O X
X O X X O O X
X X X
X X X
Depth:8
X X X
X O X
X X X X O X X
X O X X O O X
X X O O O O X
X X X
X X X
Depth:9
X X X
X X X
X X X O O X X
X O X O O O X
X X O O O O X
X X X
X X X
Depth:10
X X X
X X X
X X X O O X X
X O X O O O X
X X O O X O X
X X O
X X O
Depth:11
X X X
X X X
X O O X O X X
X O X O O O X
X X O O X O X
X X O
X X O
Depth:12
X X X
X X X
X O O X O X X
X O X O O O X
X X O O X O X
X X O
O O X
Depth:13
X X O
X X O
X O O X X X X
X O X O O O X
X X O O X O X
X X O
O O X
Depth:14
X X O
X X O
X O O X X X X
X O X O O O X
X X O O X O X
O O X
O O X
Depth:15
X X O
X X O
X O O X X X X
X O X O X O X
X X O O O O X
O O O
O O X
Depth:16
X X O
X X X
X O O X O X X
X O X O O O X
X X O O O O X
O O O
O O X
Depth:17
X X O
X X X
X O O X X O O
X O X O O O X
X X O O O O X
O O O
O O X
Depth:18
O X O
O X X
X O X X X O O
X O X O O O X
X X O O O O X
O O O
O O X
Depth:19
O X O
O X X
X O X X X O O
X O X O O O X
O O X O O O X
O O O
O O X
Depth:20
O X O
O X X
X X O O X O O
X O X O O O X
O O X O O O X
O O O
O O X
Depth:21
O O O
O O X
X X O X X O O
X O X O O O X
O O X O O O X
O O O
O O X
Depth:22
O O O
O O X
O O X X X O O
X O X O O O X
O O X O O O X
O O O
O O X
Depth:23
O O O
O O X
O X O O X O O
X O X O O O X
O O X O O O X
O O O
O O X
Depth:24
O O O
O O X
O X X O X O O
X O O O O O X
O O O O O O X
O O O
O O X
Depth:25
O O O
O O X
O X X O X O X
X O O O O O O
O O O O O O O
O O O
O O X
Depth:26
O O O
O O X
O O O X X O X
X O O O O O O
O O O O O O O
O O O
O O X
Depth:27
O O O
O O X
O O O O O X X
X O O O O O O
O O O O O O O
O O O
O O X



DFS with a Node Selection Heuristic Implementation:

The difference compared to other algorithms is that the expand algorithm is different. Children created in the Expand function are placed at the end of the frontier list in manhattan distance order.

```
def heuristic_dfs_search(frontier):
    current_depth = 0 # current search depth
    expanded_counter = 0 # counter for expanded nodes
    frontier_max = 0 # max frontier length
    start = time.time() # starting time
    best_solution_in_our_hand = initial_state # best solution for t=0
    while (True):
        #print("\n")
        #print(len(frontier))
        time_processed = time.time() - start
        if(len(frontier) > frontier_max): frontier_max = len(frontier)
        if time_processed >= time_limit:
            return best_solution_in_our_hand, time_processed,
expanded_counter, frontier_max
        state = frontier.pop(len(frontier)-1)
        if(state.map == goal_state_map):
            print("Successful!!!!")
            state.print_board()
            return state, time_processed, expanded_counter,
frontier_max
        else:
            if(state.depth == current_depth):
```

```
                state.print_board()
            else:
                #print("---->")
                #state.print_board()
                if(state.depth > best_solution_in_our_hand.depth):
                    best_solution_in_our_hand = state
                if(state.depth >= 29):
                    print("\n")
                    state.print_board()
                current_depth += 1
            expand_state_HEURISTIC3(state,frontier)
            expanded_counter += 1
```

```
def expand_state_HEURISTIC3(state,frontier):
    to_be_added = [] # children's temp list
    for hole in state.holes: # state holes
        x = hole[0] # hole coordinate for x
        y = hole[1] # hole coordinate for y
        parent = state # parent assign
        depth = int(state.depth + 1) # depth assign
        if((y!=0) and (y!=1)): # limit of map
            if ((state.map[y-1][x]== 1) and (state.map[y-2][x]==1)): #
north check
                map = copy.deepcopy(state.map)
                map[y][x] = 1
                map[y-1][x] = 0
                map[y-2][x] = 0
                new_holes = [[x,y-1],[x,y-2]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)
                holes.remove([x,y])
                to_be_added.append(State( holes, map, depth, parent))

        if((y!=5) and (y!=6)): # limit of map
            if ((state.map[y+1][x]== 1) and (state.map[y+2][x]==1)): #
south check
                map2 = copy.deepcopy(state.map)
                map2[y][x] = 1
                map2[y+1][x] = 0
                map2[y+2][x] = 0
                new_holes = [[x,y+1],[x,y+2]]
```

```
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                to_be_added.append(State( holes, map2, depth, parent))
        if((x != 0) and (x != 1)): # limit of map
            if ((state.map[y][x-1]== 1) and (state.map[y][x-2]==1)): #
west check
                map2 = copy.deepcopy(state.map)
                map2[y][x] = 1
                map2[y][x-1] = 0
                map2[y][x-2] = 0
                new_holes = [[x-1,y],[x-2,y]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                to_be_added.append(State( holes, map2, depth, parent))
        if((x != 5) and (x != 6)): # limit of map
            if ((state.map[y][x+1]== 1) and (state.map[y][x+2]==1)): #
east check
                map2 = copy.deepcopy(state.map)
                map2[y][x] = 1
                map2[y][x+1] = 0
                map2[y][x+2] = 0
                new_holes = [[x+1,y],[x+2,y]]
                holes = copy.deepcopy(state.holes)
                for ho in new_holes:
                    holes.append(ho)

                holes.remove([x,y])
                to_be_added.append(State( holes, map2, depth, parent))

    sorted_to_be_added = {} # manhattan sorted list
    for newstate in to_be_added: # for loop in to be added states
        manhattan_distance_sum = 0
        for row in range(7): # row
            for column in range(7): # column
                if(newstate.map[row][column] == 1): # if current index
is 1
```

```
                    manhattan_distance_sum += abs(row-3) +
abs(column-3) # calculation absolute value for manhattan distance
                    sorted_to_be_added[newstate] =
manhattan_distance_sum # add manhattan list as key-value
    sorted_to_be_added = dict(sorted(sorted_to_be_added.items(),
key=lambda item: item[1],reverse=True)) # sort the list
    sorted_to_be_added_only_keys = list(sorted_to_be_added.keys())
    fuc = list(sorted_to_be_added.values())
    for i_state in sorted_to_be_added_only_keys: # add to frontier in
manhattan order
            frontier.append(i_state)
```

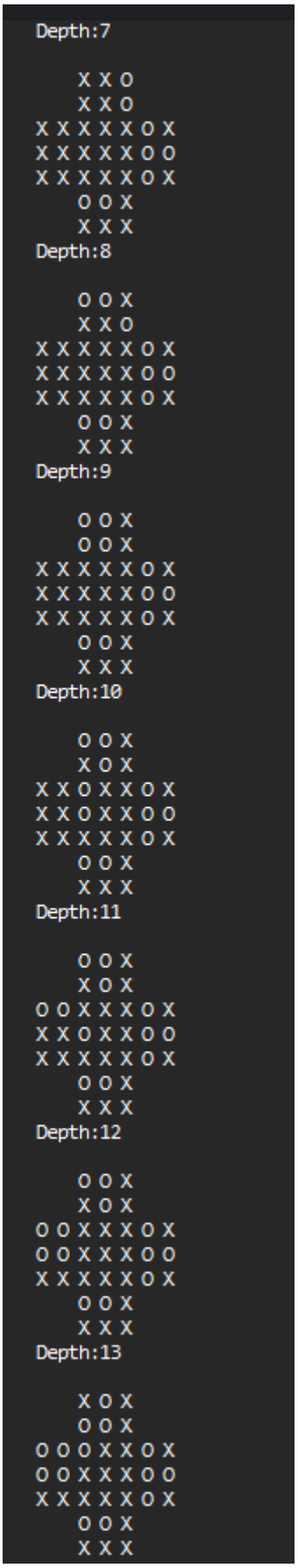
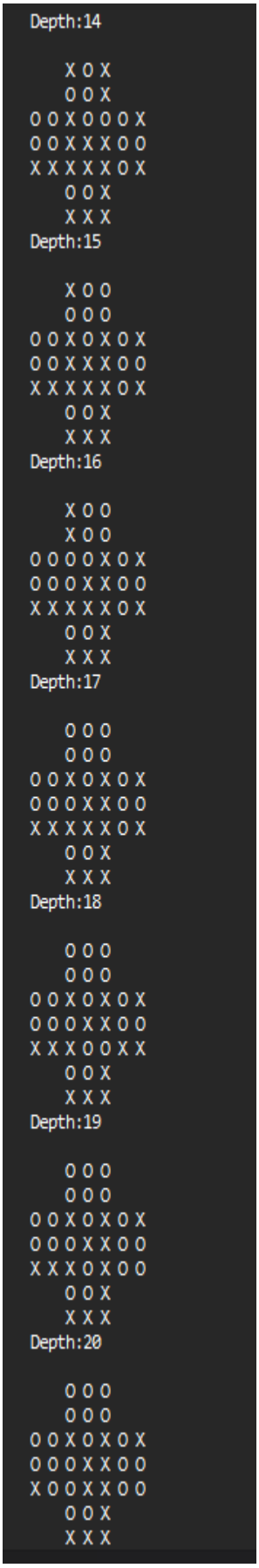
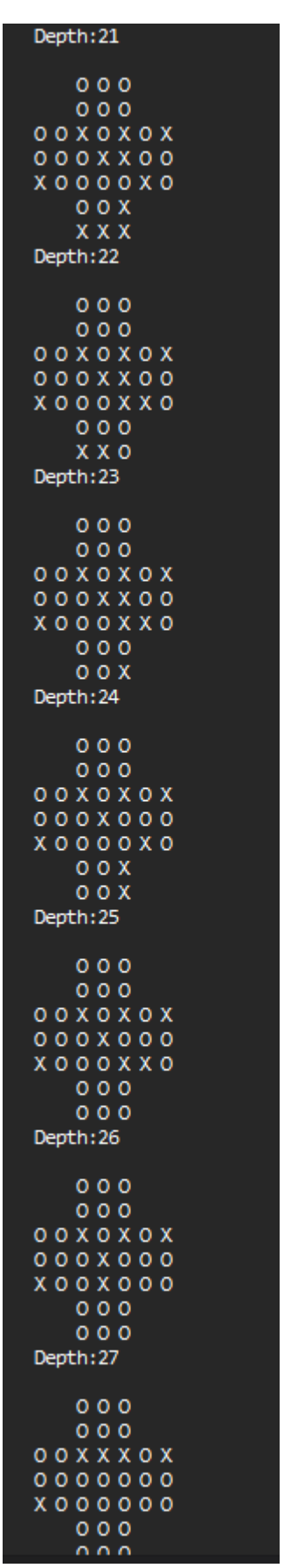
DFS with a Node Selection Heuristic Test:

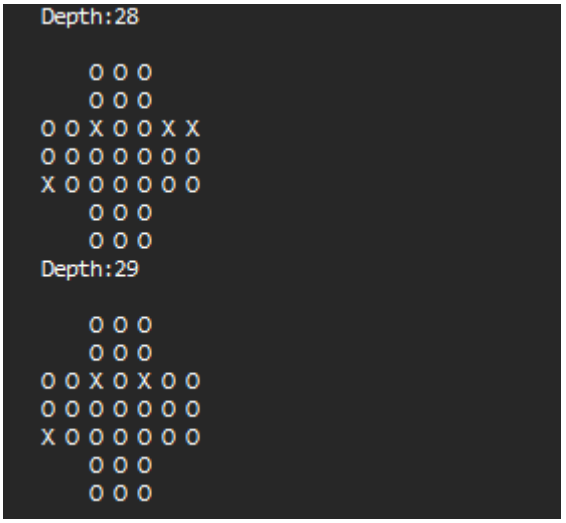
This algorithm was run for 60 minutes.

```
<<<PEG SOLITAIRE SOLVER AI>>>
-----------------------------
a. Breadth-First Search
b. Depth-First Search
c. Iterative Deepening Search
d. Depth-First Search with Random Selection
e. Depth-First Search with a Node Selection Heuristic


Select a method: e
Type your time limit (in minutes): 60
```

Result:

```
Sub-optimum Solution Found with 3 remaining pegs
Time spent: 60 minutes
41574322 nodes expanded during the search.
Maximum 162 nodes stored in the memory during the search.
```


Depth:0
X X X
X X X
X X X X X X X
X X X O X X X
X X X X X X X
X X X
X X X
Depth:1
X X X
X X X
X X X X X X X
X X X X O O X
X X X X X X X
X X X
X X X
Depth:2
X X X
X X X
X X X X X X X
X X X X X O X
X X X X O X X
X X O
X X X
Depth:3
X X X
X X X
X X X X X X X
X X X X X O X
X X X X X O O
X X O
X X X
Depth:4
X X X
X X X
X X X X X X O
X X X X X O O
X X X X X O X
X X O
X X X
Depth:5
X X X
X X X
X X X X X X O
X X X X X O O
X X X X X O X
O O X
X X X
Depth:6
X X X
X X X
X X X X O O X
X X X X X O O
X X X X X O X
O O X
X X X

Depth:7
X X O
X X O
X X X X X O X
X X X X X O O
X X X X X O X
O O X
X X X
Depth:8
O O X
X X O
X X X X X O X
X X X X X O O
X X X X X O X
O O X
X X X
Depth:9
O O X
O O X
X X X X X O X
X X X X X O O
X X X X X O X
O O X
X X X
Depth:10
O O X
X O X
X X O X X O X
X X O X X O O
X X X X X O X
O O X
X X X
Depth:11
O O X
X O X
O O X X X O X
X X O X X O O
X X X X X O X
O O X
X X X
Depth:12
O O X
X O X
O O X X X O X
O O X X X O O
X X X X X O X
O O X
X X X
Depth:13
X O X
O O X
O O O X X O X
O O X X X O O
X X X X X O X
O O X
X X X

Depth:14
X O X
O O X
O O X O O O X
O O X X X O O
X X X X X O X
O O X
X X X
Depth:15
X O O
O O O
O O X O X O X
O O X X X O O
X X X X X O X
O O X
X X X
Depth:16
X O O
X O O
O O O O X O X
O O O X X O O
X X X X X O X
O O X
X X X
Depth:17
O O O
O O O
O O X O X O X
O O O X X O O
X X X X X O X
O O X
X X X
Depth:18
O O O
O O O
O O X O X O X
O O O X X O O
X X X O O X X
O O X
X X X
Depth:19
O O O
O O O
O O X O X O X
O O O X X O O
X X X O X O O
O O X
X X X
Depth:20
O O O
O O O
O O X O X O X
O O O X X O O
X O O X X O O
O O X
X X X

Depth:21
O O O
O O O
O O X O X O X
O O O X X O O
X O O O O X O
O O X
X X X
Depth:22
O O O
O O O
O O X O X O X
O O O X X O O
X O O O X X O
O O O
X X O
Depth:23
O O O
O O O
O O X O X O X
O O O X X O O
X O O O X X O
O O O
O O X
Depth:24
O O O
O O O
O O X O X O X
O O O X O O O
X O O O O X O
O O X
O O X
Depth:25
O O O
O O O
O O X O X O X
O O O X O O O
X O O O X X O
O O O
O O O
Depth:26
O O O
O O O
O O X O X O X
O O O X O O O
X O O X O O O
O O O
O O O
Depth:27
O O O
O O O
O O X X X O X
O O O O O O O
X O O O O O O
O O O
O O O


Depth:28
O O O
O O O
O O X O O X X
O O O O O O O
X O O O O O O
O O O
O O O
Depth:29
O O O
O O O
O O X O X O O
O O O O O O O
X O O O O O O
O O O
O O O