jupyter a_star_search Last Checkpoint: 11 minutes ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[6]:
```
import heapq

class Graph:
    def __init__(self):
        self.heuristic = {}
        self.adjacency = {}

    def add_node(self, x):
        self.adjacency[x] = []

    def add_edge(self, x, y, weight):
        if x in self.adjacency and y in self.adjacency:
            self.adjacency[x].append((y, weight))
            self.adjacency[y].append((x, weight))

    def add_heuristic(self, x, h):
        self.heuristic[x] = h

    def find_target(self, s, t):
        if s not in self.adjacency or t not in self.adjacency:
            return "nodes not found"

        q = []
        heapq.heappush(q, (self.heuristic[s], 0, s))

        g_cost = {s: 0}
        came_from = {}
        came_from[s] = None

        while q:
            _, cur_cost, cur_node = heapq.heappop(q)
```

jupyter a_star_search Last Checkpoint: 12 minutes ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

```
            if cur_node == t:
                path = []
                while cur_node is not None:
                    path.append(cur_node)
                    cur_node = came_from[cur_node]
                return path[::-1]

            for neighbor, weight in self.adjacency[cur_node]:
                new_cost = cur_cost + weight

                if neighbor not in g_cost or new_cost < g_cost[neighbor]:
                    g_cost[neighbor] = new_cost
                    f_cost = new_cost + self.heuristic[neighbor]
                    came_from[neighbor] = cur_node
                    heapq.heappush(q, (f_cost, new_cost, neighbor))

        return "goal not reachable"
```

[7]:
```
g = Graph()
g.add_node('oradea')
g.add_node('zerind')
g.add_node('arad')
g.add_node('sibiu')
g.add_node('timisoara')
g.add_node('lugoj')
g.add_node('fagaras')
g.add_node('mehadia')
g.add_node('drobeta')
g.add_node('rimnicu')
g.add_node('craiova')
g.add_node('pitesti')
g.add_node('bucharest')
g.add_node('giurgiu')
```

jupyter a_star_search Last Checkpoint: 12 minutes ago

File Edit View Run Kernel Settings Help Not Tr

Code JupyterLab Python 3 (ipykern

[8]:
```
g.add_edge('oradea', 'zerind', 71)
g.add_edge('oradea', 'sibiu', 151)
g.add_edge('zerind', 'arad', 75)
g.add_edge('arad', 'sibiu', 140)
g.add_edge('arad', 'timisoara', 118)
g.add_edge('sibiu', 'fagaras', 99)
g.add_edge('sibiu', 'rimnicu', 80)
g.add_edge('timisoara', 'lugoj', 111)
g.add_edge('lugoj', 'mehadia', 70)
g.add_edge('mehadia', 'drobeta', 75)
g.add_edge('drobeta', 'craiova', 120)
g.add_edge('fagaras', 'bucharest', 211)
g.add_edge('rimnicu', 'pitesti', 97)
g.add_edge('rimnicu', 'craiova', 146)
g.add_edge('craiova', 'pitesti', 138)
g.add_edge('pitesti', 'bucharest', 101)
g.add_edge('bucharest', 'giurgiu', 90)
```

[11]:
```
g.add_heuristic('arad', 366)
g.add_heuristic('bucharest', 0)
g.add_heuristic('craiova', 160)
g.add_heuristic('drobeta', 242)
g.add_heuristic('fagaras', 176)
g.add_heuristic('giurgiu', 77)
g.add_heuristic('lugoj', 244)
g.add_heuristic('mehadia', 241)
g.add_heuristic('oradea', 380)
g.add_heuristic('pitesti', 100)
g.add_heuristic('rimnicu', 193)
g.add_heuristic('sibiu', 253)
g.add_heuristic('timisoara', 329)
g.add_heuristic('zerind', 374)
```

jupyter a_star_search Last Checkpoint: 13 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab P


```
g.add_edge('craiova', 'pitesti', 138)
g.add_edge('pitesti', 'bucharest', 101)
g.add_edge('bucharest', 'giurgiu', 90)
```

[11]:
```
g.add_heuristic('arad', 366)
g.add_heuristic('bucharest', 0)
g.add_heuristic('craiova', 160)
g.add_heuristic('drobeta', 242)
g.add_heuristic('fagaras', 176)
g.add_heuristic('giurgiu', 77)
g.add_heuristic('lugoj', 244)
g.add_heuristic('mehadia', 241)
g.add_heuristic('oradea', 380)
g.add_heuristic('pitesti', 100)
g.add_heuristic('rimnicu', 193)
g.add_heuristic('sibiu', 253)
g.add_heuristic('timisoara', 329)
g.add_heuristic('zerind', 374)
```


[12]:
```
start_node = 'arad'
goal_node = 'bucharest'

path = g.find_target(start_node, goal_node)
print("Path from", start_node, "to", goal_node, ":", path)
```

```
Path from arad to bucharest : ['arad', 'sibiu', 'rimnicu', 'pitesti', 'bucharest']
```

jupyter alpha_beta_tree Last Checkpoint: 14 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab

```python
import math
def alpha_beta(node, depth, alpha, beta, is_max):
    if depth == 3:  # Leaf node
        return node  # Return the value at the leaf node
    if is_max:  # Max's turn
        value = -math.inf
        for child in node:
            value = max(value, alpha_beta(child, depth + 1, alpha, beta, False))
            alpha = max(alpha, value)
            if alpha >= beta:  # Beta cut-off
                break
        return value
    else:  # Min's turn
        value = math.inf
        for child in node:
            value = min(value, alpha_beta(child, depth + 1, alpha, beta, True))
            beta = min(beta, value)
            if alpha >= beta:  # Alpha cut-off
                break
        return value

# Leaf values from the tree
leaf_values = [
    [[6, 5], [2, 6]],
    [[4, 7], [2, 2]],
    [[5, 1], [1, 5]],
    [[3, 9], [2, 6]]
]
# Compute optimal value
result = alpha_beta(leaf_values, 0, -math.inf, math.inf, True)
print("Optimal value:", result)
```

```
Optimal value: 6
```

jupyter banana Last Checkpoint: 15 minutes ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

•[2]:

```python
from collections import deque

def solve_monkey_and_banana():
    # Initial state: (monkey_position, box_position, monkey_on_box, monkey_has_banana)
    start_state = ('door', 'window', 'no', 'no')
    # Goal condition: (any, any, any, 'yes')
    goal_condition = lambda state: state[3] == 'yes'
    # BFS queue: stores (current state, path taken)
    queue = deque([(start_state, [start_state])])

    # Set to store visited states
    visited = set([start_state])
    # BFS loop
    while queue:
        current_state, path = queue.popleft()

        # Check if we have reached the goal
        if goal_condition(current_state):
            return path

        # Generate next possible states
        next_states = generate_next_states(current_state)

        for next_state in next_states:
            if next_state not in visited:
                visited.add(next_state)
                queue.append((next_state, path + [next_state]))
    # No solution found
    return None

def generate_next_states(state):
    monkey_position, box_position, monkey_on_box, monkey_has_banana = state
```

jupyter banana Last Checkpoint: 16 minutes ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

```python
def generate_next_states(state):
    monkey_position, box_position, monkey_on_box, monkey_has_banana = state
    next_states = []

    # Action: Walk (P)
    if monkey_on_box == 'no':  # Monkey can only walk if it's not on the box
        possible_positions = ['door', 'window', 'middle']
        for pos in possible_positions:
            if pos != monkey_position:
                next_states.append((pos, box_position, 'no', monkey_has_banana))

    # Action: Push (P)
    if monkey_on_box == 'no' and monkey_position == box_position:
        possible_positions = ['door', 'window', 'middle']
        for pos in possible_positions:
            if pos != monkey_position:
                next_states.append((pos, pos, 'no', monkey_has_banana))

    # Action: Climb
    if monkey_position == box_position and monkey_on_box == 'no':
        next_states.append((monkey_position, box_position, 'yes', monkey_has_banana))

    # Action: Grasp
    if monkey_position == 'middle' and box_position == 'middle' and monkey_on_box == 'yes' and monkey_has_banana == 'no':
        next_states.append(('middle', 'middle', 'yes', 'yes'))

    return next_states
```

jupyter banana Last Checkpoint: 16 minutes ago

File Edit View Run Kernel Settings Help

Code


```python
        next_states.append((monkey_position, box_position, 'yes', monkey_has_banana))

    # Action: Grasp
    if monkey_position == 'middle' and box_position == 'middle' and monkey_on_box == 'yes' and monkey_has_banana == 'no'
        next_states.append(('middle', 'middle', 'yes', 'yes'))

    return next_states
```


[3]:

```python
def print_solution(solution):
    if solution:
        print("Solution found:")
        for state in solution:
            print(f"Monkey: {state[0]}, Box: {state[1]}, On Box: {state[2]}, Has Banana: {state[3]}")
    else:
        print("No solution found.")

# Run the solver
solution = solve_monkey_and_banana()
print_solution(solution)
```

```
Solution found:
Monkey: door, Box: window, On Box: no, Has Banana: no
Monkey: window, Box: window, On Box: no, Has Banana: no
Monkey: middle, Box: middle, On Box: no, Has Banana: no
Monkey: middle, Box: middle, On Box: yes, Has Banana: no
Monkey: middle, Box: middle, On Box: yes, Has Banana: yes
```

jupyter BFS Last Checkpoint: 42 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab Python 3 (ipyke

[22]:
```python
##ADJACENCY LIST
from collections import defaultdict
from collections import deque

class graph:
    def __init__(self):
        self.adjacent=defaultdict(list)
        self.parent=defaultdict()
    def add_edges(self,f,t):
        self.adjacent[f].append(t)
    def bfs(self,s,g):
        visited=defaultdict(bool)
        q=deque()
        q.append(s)
        while q:
            ele=q[0]
            q.popleft()
            visited[ele]=True
            for child in self.adjacent[ele]:
                if child not in visited:
                    self.parent[child]=ele
                    if child==g:
                        print("shortest path is")
                        self.printpath(child)
                        return
                    q.append(child)
                    visited[child]=True
        print("target element not found")
    def printpath(self,c):
        if c in self.parent:
            self.printpath(self.parent[c])
            print(c,"-->",end=" ")
        else:
            print(c,"-->",end=" ")
```

jupyter BFS Last Checkpoint: 42 minutes ago

File Edit View Run Kernel Settings Help

Not Trusted

Code JupyterLab Python 3 (ipykernel)


```python
    def printpath(self,c):
        if c in self.parent:
            self.printpath(self.parent[c])
            print(c,"-->",end=" ")
        else:
            print(c,"-->",end=" ")
```


[27]:
```python
if __name__=='__main__':
    g=graph()
    g.add_edges("A","F")
    g.add_edges("A","C")
    g.add_edges("A","B")
    g.add_edges("B","C")
    g.add_edges("B","A")
    g.add_edges("B","G")
    g.add_edges("C","A")
    g.add_edges("C","B")
    g.add_edges("C","D")
    g.add_edges("C","E")
    g.add_edges("C","F")
    g.add_edges("C","G")
    g.add_edges("D","C")
    g.add_edges("D","F")
    g.add_edges("D","E")
    g.add_edges("D","J")
    g.add_edges("E","C")
    g.add_edges("E","D")
    g.add_edges("E","G")
    g.add_edges("E","J")
    g.add_edges("E","K")
    g.add_edges("F","A")
    g.add_edges("F","C")
    g.add_edges("F","D")
    g.add_edges("G","B")
```

jupyter BFS Last Checkpoint: 43 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterL

```python
g.add_edges("B","A")
g.add_edges("B","G")
g.add_edges("C","A")
g.add_edges("C","B")
g.add_edges("C","D")
g.add_edges("C","E")
g.add_edges("C","F")
g.add_edges("C","G")
g.add_edges("D","C")
g.add_edges("D","F")
g.add_edges("D","E")
g.add_edges("D","J")
g.add_edges("E","C")
g.add_edges("E","D")
g.add_edges("E","G")
g.add_edges("E","J")
g.add_edges("E","K")
g.add_edges("F","A")
g.add_edges("F","C")
g.add_edges("F","D")
g.add_edges("G","B")
g.add_edges("G","C")
g.add_edges("G","E")
g.add_edges("G","K")
g.add_edges("J","D")
g.add_edges("J","E")
g.add_edges("J","K")
g.add_edges("K","E")
g.add_edges("K","G")
g.add_edges("K","J")
g.bfs("A","K")
```

```
shortest path is
A --> C --> E --> K -->
```

jupyter boat Last Checkpoint: 43 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab

[2]:

```python
def solve_missionaries_and_cannibals():
    from collections import deque
    initial_state = (3, 3, 'L')
    goal_state = (0, 0, 'R')

    queue = deque([(initial_state, [initial_state])])

    visited = set([initial_state])

    while queue:
        (m, c, boat), path = queue.popleft()

        # Check if we have reached the goal
        if (m, c, boat) == goal_state:
            return path

        # Generate next possible states
        for new_state in generate_valid_moves(m, c, boat):
            if new_state not in visited:
                visited.add(new_state)
                queue.append((new_state, path + [new_state]))

    # No solution found
    return None

def generate_valid_moves(m, c, boat):
    moves = []
    if boat == 'L':
        possible_moves = [(m-2, c, 'R'), (m, c-2, 'R'), (m-1, c-1, 'R'), (m-1, c, 'R'), (m, c-1, 'R')]
    else:
        possible_moves = [(m+2, c, 'L'), (m, c+2, 'L'), (m+1, c+1, 'L'), (m+1, c, 'L'), (m, c+1, 'L')]
```

jupyter boat Last Checkpoint: 44 minutes ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

```python
    else:
        possible_moves = [(m+2, c, 'L'), (m, c+2, 'L'), (m+1, c+1, 'L'), (m+1, c, 'L'), (m, c+1, 'L')]

    # Check for valid states
    for new_m, new_c, new_boat in possible_moves:
        if 0 <= new_m <= 3 and 0 <= new_c <= 3:
            if new_m == 0 or new_m >= new_c:
                if (3 - new_m) == 0 or (3 - new_m) >= (3 - new_c):

                    moves.append((new_m, new_c, new_boat))

    return moves
```

[3]:
```python
solution = solve_missionaries_and_cannibals()
if solution:
    print("Solution path:")
    for state in solution:
        print(state)
else:
    print("No solution found.")
```

```
Solution path:
(3, 3, 'L')
(3, 1, 'R')
(3, 2, 'L')
(3, 0, 'R')
(3, 1, 'L')
(1, 1, 'R')
(2, 2, 'L')
(0, 2, 'R')
(0, 3, 'L')
(0, 1, 'R')
(1, 1, 'L')
(0, 0, 'R')
```

jupyter cat_FOL_lab Last Checkpoint: 44 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab

[1]:
```python
class Constant:
    def __init__(self, name):
        self.name = str(name)

    def __repr__(self):
        return self.name

class Predicate:
    def __init__(self, name, *args):
        self.name = name
        self.args = args

    def __repr__(self):
        return f"{self.name}({', '.join(map(str, self.args))})"

class Quantifier:
    def __init__(self, quantifier, variable, statement):
        self.quantifier = quantifier
        self.variable = variable
        self.statement = statement

    def __repr__(self):
        return f"{self.quantifier}{self.variable} ({self.statement})"

class LogicalConnective:
    def __init__(self, connective, left, right):
        self.connective = connective
        self.left = left
        self.right = right

    def __repr__(self):
        return f"({self.left} {self.connective} {self.right})"
```

jupyter cat_FOL_lab Last Checkpoint: 45 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab Python

```python
    def __repr__(self):
        return f"({self.left} {self.connective} {self.right})"


Jack = Constant("Jack")
Curiosity = Constant("Curiosity")
Tuna = Constant("Tuna")

def animal(y):
    return Predicate("Animal", y)

def loves(x, y):
    return Predicate("Loves", x, y)

def kills(x, y):
    return Predicate("Kills", x, y)


everyone_loves_animals = Quantifier("∀", "x",
    LogicalConnective("→",
        Quantifier("∀", "y", LogicalConnective("→", animal(Constant("y")), loves(Constant("x"), Constant("y")))),
        Quantifier("∃", "z", loves(Constant("z"), Constant("x")))
    )
)


anyone_kills_animal = Quantifier("∀", "x",
    LogicalConnective("→",
        Quantifier("∃", "y", LogicalConnective("∧", animal(Constant("y")), kills(Constant("x"), Constant("y")))),
        Quantifier("∀", "z", Predicate("¬", loves(Constant("z"), Constant("x"))))
    )
)

jack_loves_all_animals = Quantifier("∀", "y", loves(Jack, Constant("y")))
```

jupyter cat_FOL_lab Last Checkpoint: 46 minutes ago

File Edit View Run Kernel Settings Help

Not Trusted

Code JupyterLab Python 3 (ipykernel)

```python
jack_loves_all_animals = Quantifier("∀", "y", loves(Jack, Constant("y")))

either_jack_or_curiosity_killed = LogicalConnective("∨", kills(Jack, Tuna), kills(Curiosity, Tuna))

did_curiosity_kill_cat = Predicate('¬',kills(Curiosity, Tuna))

tunacatis_animal=Quantifier('∀','x',LogicalConnective('→',Predicate('Cat',Constant('x')),animal(Constant('x'))))

print("1. Everyone who loves all animals is loved by someone:\n", everyone_loves_animals)
print("2. Anyone who kills an animal is loved by no one:\n", anyone_kills_animal)
print("3. Jack loves all animals:\n", jack_loves_all_animals)
print("4. Either Jack or Curiosity killed the cat named Tuna:\n", either_jack_or_curiosity_killed)
print("5. Did Curiosity kill the cat?\n", did_curiosity_kill_cat)
print("6",Predicate('Cat',Constant('Tuna')))
print("7",tunacatis_animal)
```

```
1. Everyone who loves all animals is loved by someone:
 ∀x ((∀y ((Animal(y) → Loves(x, y))) → ∃z (Loves(z, x))))
2. Anyone who kills an animal is loved by no one:
 ∀x ((∃y ((Animal(y) ∧ Kills(x, y))) → ∀z (¬(Loves(z, x)))))
3. Jack loves all animals:
 ∀y (Loves(Jack, y))
4. Either Jack or Curiosity killed the cat named Tuna:
 (Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna))
5. Did Curiosity kill the cat?
 ¬(Kills(Curiosity, Tuna))
6 Cat(Tuna)
7 ∀x ((Cat(x) → Animal(x)))
```

jupyter Cryptarithmetic Last Checkpoint: 46 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab Python

[1]:
```python
def find_value(word, assigned):
    return int("".join(str(assigned[char]) for char in word))

def is_valid_assignment(word1, word2, result, assigned):
    return all(assigned[word[0]] != 0 for word in (word1, word2, result))

def solve_recursively(word1, word2, result, letters, assigned, solutions):
    if not letters:
        if is_valid_assignment(word1, word2, result, assigned):
            num1, num2, num_res = find_value(word1, assigned), find_value(word2, assigned), find_value(result, assigned)
            if num1 + num2 == num_res:
                solutions.append((f'{num1} + {num2} = {num_res}', assigned.copy()))
        return

    letter = letters.pop()
    for num in range(10):
        if num not in assigned.values():
            assigned[letter] = num
            solve_recursively(word1, word2, result, letters, assigned, solutions)
            assigned.pop(letter)
    letters.append(letter)
```

[2]:
```python
def solve(word1, word2, result):
    letters = list(set(word1 + word2 + result))
    if len(result) > max(len(word1), len(word2)) + 1 or len(letters) > 10:
        print('0 Solutions!')
        return

    solutions = []
    solve_recursively(word1, word2, result, letters, {}, solutions)
    if solutions:
        print('\nSolutions:')
```

jupyter Cryptarithmetic Last Checkpoint: 47 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab

[2]:
```python
def solve(word1, word2, result):
    letters = list(set(word1 + word2 + result))
    if len(result) > max(len(word1), len(word2)) + 1 or len(letters) > 10:
        print('0 Solutions!')
        return

    solutions = []
    solve_recursively(word1, word2, result, letters, {}, solutions)
    if solutions:
        print('\nSolutions:')
        for equation, mapping in solutions:
            print(f'{equation}\t{mapping}')
```

[3]:
```python
print('CRYPTARITHMETIC PUZZLE SOLVER')
print('WORD1 + WORD2 = RESULT')
word1 = input('Enter WORD1: ').upper()
word2 = input('Enter WORD2: ').upper()
result = input('Enter RESULT: ').upper()

if not all(w.isalpha() for w in (word1, word2, result)):
    raise ValueError('Inputs should only consist of alphabets.')

solve(word1, word2, result)
```

```
CRYPTARITHMETIC PUZZLE SOLVER
WORD1 + WORD2 = RESULT
Enter WORD1: SEND
Enter WORD2: MORE
Enter RESULT: MONEY

Solutions:
9567 + 1085 = 10652     {'O': 0, 'R': 8, 'M': 1, 'E': 5, 'Y': 2, 'N': 6, 'S': 9, 'D': 7}
```

jupyter depth_limited Last Checkpoint: 48 minutes ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[1]:
```python
def dls(graph, start, goal, depth, visited=None, parents=None):
    if parents is None:
        parents = {start: None}
    if visited is None:
        visited = set()
    if depth < 0:
        return None
    if start == goal:
        return construct_path(start, parents)

    visited.add(start)

    for neighbor in graph[start]:
        if neighbor not in visited:
            parents[neighbor] = start
            result = dls(graph, neighbor, goal, depth - 1, visited, parents)
            if result is not None:
                return result
    return None
```

[2]:
```python
def construct_path(goal, parents):
    path = []
    while goal is not None:
        path.append(goal)
        goal = parents[goal]
    path.reverse()
    return path
```

jupyter depth_limited Last Checkpoint: 48 minutes ago

File Edit View Run Kernel Settings Help

Code

[2]:
```python
def construct_path(goal, parents):
    path = []
    while goal is not None:
        path.append(goal)
        goal = parents[goal]
    path.reverse()
    return path
```


[11]:
```python
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': [],
    'E': [],
    'F': [],
    'G': ['H'],
    'H': []
}

start = 'A'
goal = 'F'
depth = 2
```

[12]:
```python
path = dls(graph, start, goal, depth)
if path:
    print("Path found:", path)
else:
    print("Not Found")
```

```
Path found: ['A', 'C', 'F']
```

jupyter DFS Last Checkpoint: 48 minutes ago

File Edit View Run Kernel Settings Help | Not Trusted

Code | JupyterLab | Python 3 (ipykernel)

[1]:
```python
from collections import deque

def dfs(graph, initial, goal):
    stack = [initial]
    explored = set()
    parents = {initial: None}

    while stack:
        node = stack.pop()
        if node not in explored:
            explored.add(node)
            if goal_test(node):
                return construct_path(node, parents)
            for neighbor in graph[node]:
                if neighbor not in explored:
                    parents[neighbor] = node
                    stack.append(neighbor)
    return None

def construct_path(node, parents):
    path = []
    current = node
    while current is not None:
        path.append(current)
        current = parents[current]
    path.reverse()
    return path

if __name__ == "__main__":
    graph = {
        'A': ['F', 'C', 'B'],
        'B': ['A', 'C', 'G'],
```

jupyter DFS Last Checkpoint: 49 minutes ago

File Edit View Run Kernel Settings Help

Code | Jupyte

```python
def construct_path(node, parents):
    path = []
    current = node
    while current is not None:
        path.append(current)
        current = parents[current]
    path.reverse()
    return path

if __name__ == "__main__":
    graph = {
        'A': ['F', 'C', 'B'],
        'B': ['A', 'C', 'G'],
        'C': ['A', 'B', 'D', 'E', 'F', 'G'],
        'D': ['C', 'F', 'E', 'J'],
        'E': ['C', 'D', 'G', 'J', 'K'],
        'F': ['A', 'C', 'D'],
        'G': ['B', 'C', 'E', 'K'],
        'J': ['D', 'E', 'K'],
        'K': ['E', 'G', 'J']
    }

    initial_state = 'A'
    goal_state = 'K'

    def goal_test(state):
        return state == goal_state

    solution = dfs(graph, initial_state, goal_test)
    if solution:
        print(f"Solution path found: {solution}")
    else:
        print("No solution found.")
```

jupyter DFS Last Checkpoint: 49 minutes ago

File Edit View Run Kernel Settings Help

Code

```
        'B': ['A', 'C', 'G'],
        'C': ['A', 'B', 'D', 'E', 'F', 'G'],
        'D': ['C', 'F', 'E', 'J'],
        'E': ['C', 'D', 'G', 'J', 'K'],
        'F': ['A', 'C', 'D'],
        'G': ['B', 'C', 'E', 'K'],
        'J': ['D', 'E', 'K'],
        'K': ['E', 'G', 'J']
    }

    initial_state = 'A'
    goal_state = 'K'

    def goal_test(state):
        return state == goal_state

    solution = dfs(graph, initial_state, goal_test)
    if solution:
        print(f"Solution path found: {solution}")
    else:
        print("No solution found.")
```

```
Solution path found: ['A', 'B', 'G', 'K']
```

[ ]:

jupyter FOL Last Checkpoint: 50 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab

[7]:

```
import pandas as pd
import numpy as np
class Constant:
    def __init__(self, name):
        self.name = name
    def __repr__(self):
        return self.name

class Variable:
    def __init__(self, name):
        self.name = name
    def __repr__(self):
        return self.name

class Predicate:
    def __init__(self, name, *args):
        self.name = name
        self.args = args
    def __repr__(self):
        return f"{self.name}({','.join(map(str,self.args))})"

class Function:
    def __init__(self, name, *args):
        self.name = name
        self.args = args
    def __repr__(self):
            return f"{self.name}({', '.join(map(str, self.args))})"


class LogicalExpression:
    def __init__(self, connective, *args):
        self.connective = connective
```

jupyter FOL Last Checkpoint: 51 minutes ago

File Edit View Run Kernel Settings Help

Code Jup

```python
class LogicalExpression:
    def __init__(self, connective, *args):
        self.connective = connective
        self.args = args

    def __repr__(self):
        return f"({f' {self.connective} '.join(map(str, self.args))})"
```

[11]:

```python
if __name__ == "__main__":
    king_john = Constant("KingJohn")
    richard = Constant("Richard")
    two = Constant("2")

    X = Variable("x")
    Y = Variable("y")
    brother = Predicate("Brother", king_john, richard)
    greater_than = Predicate("GreaterThan", two, X)

    sqrt = Function("Sqrt", two)
    left_leg_of = Function("LeftLegOf", king_john)
    expression1 = LogicalExpression("^", brother, greater_than)
    expression1 = LogicalExpression("^", brother, greater_than)
    expression2 = LogicalExpression("_", left_leg_of, X)

    # Print results
    print("Constants:")
    print(king_john, richard, two)

    print("Variables:")
    print(X, Y)

    print("Predicates:")
    print(brother)
    print(greater_than)
```

jupyter FOL Last Checkpoint: 51 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab


```python
    print(king_john, richard, two)

    print("Variables:")
    print(X, Y)

    print("Predicates:")
    print(brother)
    print(greater_than)
```


```python
    print("Functions:")
    print(sqrt)
    print(left_leg_of)

    print("Logical Expressions:")
    print(expression1)
    print(expression2)
```

```
Constants:
KingJohn Richard 2
Variables:
x y
Predicates:
Brother(KingJohn,Richard)
GreaterThan(2,x)
Functions:
Sqrt(2)
LeftLegOf(KingJohn)
Logical Expressions:
(Brother(KingJohn,Richard) ^ GreaterThan(2,x))
(LeftLegOf(KingJohn) _ x)
```

[ ]:

jupyter goat_tiger Last Checkpoint: 52 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab

[1]:
```python
from collections import deque

def is_valid(state):
    boat, cabbage, goat, wolf = state

    # Check if goat is left alone with the wolf or the cabbage
    if goat == wolf and goat != boat:  # Goat alone with wolf
        return False
    if goat == cabbage and goat != boat:  # Goat alone with cabbage
        return False
    return True

def get_next_states(state):
    boat, cabbage, goat, wolf = state
    next_states = []

    # Generate possible moves
    possible_moves = [
        (1 - boat, cabbage, goat, wolf),  # Move yourself alone
        (1 - boat, 1 - cabbage, goat, wolf) if cabbage == boat else None,  # Move with cabbage
        (1 - boat, cabbage, 1 - goat, wolf) if goat == boat else None,  # Move with goat
        (1 - boat, cabbage, goat, 1 - wolf) if wolf == boat else None  # Move with wolf
    ]

    # Filter valid moves
    for move in possible_moves:
        if move and is_valid(move):
            next_states.append(move)

    return next_states

def solve_wolf_goat_cabbage():
```

jupyter goat_tiger Last Checkpoint: 53 minutes ago

File Edit View Run Kernel Settings Help Not Tr

Code JupyterLab Python 3 (ipyker

```python
def solve_wolf_goat_cabbage():

    start_state = (1, 1, 1, 1)
    goal_state = (0, 0, 0, 0)
    queue = deque([(start_state, [start_state])])

    visited = set([start_state])

    # BFS Loop
    while queue:
        current_state, path = queue.popleft()

        # Check if goal is reached
        if current_state == goal_state:
            return path

        # Get valid next states
        for next_state in get_next_states(current_state):
            if next_state not in visited:
                visited.add(next_state)
                queue.append((next_state, path + [next_state]))

    # No solution found
    return None
```

[2]:
```python
def print_solution(solution):
    if solution:
        print("Solution found:")
        for state in solution:
            print(f"Boat: {'Left' if state[0] else 'Right'}, "
                  f"Cabbage: {'Left' if state[1] else 'Right'}, "
                  f"Goat: {'Left' if state[2] else 'Right'}, "
                  f"Wolf: {'Left' if state[3] else 'Right'}")
    else:
```

jupyter goat_tiger Last Checkpoint: 53 minutes ago

File Edit View Run Kernel Settings Help

Code

[2]:
```python
def print_solution(solution):
    if solution:
        print("Solution found:")
        for state in solution:
            print(f"Boat: {'Left' if state[0] else 'Right'}, "
                  f"Cabbage: {'Left' if state[1] else 'Right'}, "
                  f"Goat: {'Left' if state[2] else 'Right'}, "
                  f"Wolf: {'Left' if state[3] else 'Right'}")
    else:
        print("No solution found.")

# Run the solver
solution = solve_wolf_goat_cabbage()
print_solution(solution)
```

```
Solution found:
Boat: Left, Cabbage: Left, Goat: Left, Wolf: Left
Boat: Right, Cabbage: Left, Goat: Right, Wolf: Left
Boat: Left, Cabbage: Left, Goat: Right, Wolf: Left
Boat: Right, Cabbage: Right, Goat: Right, Wolf: Left
Boat: Left, Cabbage: Right, Goat: Left, Wolf: Left
Boat: Right, Cabbage: Right, Goat: Left, Wolf: Right
Boat: Left, Cabbage: Right, Goat: Left, Wolf: Right
Boat: Right, Cabbage: Right, Goat: Right, Wolf: Right
```

[ ]:

jupyter greedy_best_first_serach Last Checkpoint: 54 minutes ago

File Edit View Run Kernel Settings Help

Code JupyterLab Python 3 (i

[1]:
```python
import heapq
```

[16]:
```python
class graph:
    def __init__(self):
        self.heuristic={}
        self.adjacency={}

    def add_node(self,x):
        self.adjacency[x]=[]

    def add_edge(self,x,y,weight):
        if x in self.adjacency and y in self.adjacency:
            self.adjacency[x].append((y,weight))
            self.adjacency[y].append((x,weight))

    def add_heuristic(self,x,h):
        self.heuristic[x]=h

    def find_target(self,s,t):
        if s not in self.adjacency or t not in self.adjacency:
            return "nodes not found"
        q=[]
        heapq.heappush(q,(self.heuristic[s],s))
        came_from={}
        came_from[s]=None ##also act as visited here

        while q:
            _,cur_node=heapq.heappop(q)

            if cur_node==t:
                path=[]
                while cur_node is not None:
```

jupyter greedy_best_first_serach Last Checkpoint: 55 minutes ago

File Edit View Run Kernel Settings Help

Code Jupy

```python
    def find_target(self,s,t):
        if s not in self.adjacency or t not in self.adjacency:
            return "nodes not found"
        q=[]
        heapq.heappush(q,(self.heuristic[s],s))
        came_from={}
        came_from[s]=None ##also act as visited here

        while q:
            _,cur_node=heapq.heappop(q)

            if cur_node==t:
                path=[]
                while cur_node is not None:
                    path.append(cur_node)
                    cur_node=came_from[cur_node]
                return path[::-1]

            for neighbour,weight in self.adjacency[cur_node]:
                if neighbour not in came_from:
                    came_from[neighbour]=cur_node
                    heapq.heappush(q,(self.heuristic[neighbour],neighbour))
        return "goal not reachable"
```

[17]:
```python
g = graph()
g.add_node('S')
g.add_node('A')
g.add_node('B')
g.add_node('C')
g.add_node('D')
g.add_node('E')
g.add_node('F')
g.add_node('G')
```

jupyter greedy_best_first_serach Last Checkpoint: 55 minutes ago

File Edit View Run Kernel Settings Help

Code Jupyt

[17]:
```python
g = graph()
g.add_node('S')
g.add_node('A')
g.add_node('B')
g.add_node('C')
g.add_node('D')
g.add_node('E')
g.add_node('F')
g.add_node('G')
g.add_node('H')
g.add_node('I')
```

[18]:
```python
g.add_edge('S','A',3)
g.add_edge('S','B',2)
g.add_edge('A','C',4)
g.add_edge('A','D',1)
g.add_edge('B','E',3)
g.add_edge('B','F',1)
g.add_edge('E','H',5)
g.add_edge('F','I',2)
g.add_edge('F','G',3)
```

[19]:
```python
g.add_heuristic('A', 12)
g.add_heuristic('B', 4)
g.add_heuristic('C', 7)
g.add_heuristic('D', 3)
g.add_heuristic('E', 8)
g.add_heuristic('F', 2)
g.add_heuristic('H', 4)
g.add_heuristic('I', 9)
g.add_heuristic('S', 13)
g.add_heuristic('G', 0)
```

jupyter greedy_best_first_serach Last Checkpoint: 55 minutes ago

File Edit View Run Kernel Settings Help

Code

```python
g.add_edge('A','C',4)
g.add_edge('A','D',1)
g.add_edge('B','E',3)
g.add_edge('B','F',1)
g.add_edge('E','H',5)
g.add_edge('F','I',2)
g.add_edge('F','G',3)
```

[19]:

```python
g.add_heuristic('A', 12)
g.add_heuristic('B', 4)
g.add_heuristic('C', 7)
g.add_heuristic('D', 3)
g.add_heuristic('E', 8)
g.add_heuristic('F', 2)
g.add_heuristic('H', 4)
g.add_heuristic('I', 9)
g.add_heuristic('S', 13)
g.add_heuristic('G', 0)
```

[20]:

```python
start_node = 'S'
goal_node = 'G'

path = g.find_target(start_node, goal_node)
print("Path from", start_node, "to", goal_node, ":", path)
```

```
Path from S to G : ['S', 'B', 'F', 'G']
```

jupyter iterative_deepening_depth_limited_search Last Checkpoint: 57 minutes ago

File Edit View Run Kernel Settings Help

Code

[ ]:

```python
from collections import deque
MOVES = {
    'UP': -3, 'DOWN': 3, 'LEFT': -1, 'RIGHT': 1
}

def index_to_pos(index):
    return (index // 3, index % 3)
```

[ ]:

```python
def dls(state, goal, depth, visited=None, parents=None):
    if parents is None:
        parents = {state: None}
    if visited is None:
        visited = set()
    if depth < 0:
        return None
    if state == goal:
        return construct_path(goal, parents)

    visited.add(state)

    for neighbor in get_neighbors(state):
        if neighbor not in visited:
            parents[neighbor] = state
            result = dls(neighbor, goal, depth - 1, visited, parents)
            if result is not None:
                return result

    return None
```

jupyter iterative_deepening_depth_limited_search Last Checkpoint: 58 minutes ago

File Edit View Run Kernel Settings Help

Code

[ ]:
```python
def get_neighbors(state):
    neighbors = []
    zero_pos = state.index(0)
    zero_row, zero_col = index_to_pos(zero_pos)

    for move, offset in MOVES.items():
        new_pos = zero_pos + offset

        if move == 'UP' and zero_row == 0 or \
           move == 'DOWN' and zero_row == 2 or \
           move == 'LEFT' and zero_col == 0 or \
           move == 'RIGHT' and zero_col == 2:
            continue

        if 0 <= new_pos < 9 and (move == 'LEFT' and zero_pos % 3 > 0 or \
                                 move == 'RIGHT' and zero_pos % 3 < 2 or \
                                 move in {'UP', 'DOWN'}):
            new_state = list(state)
            new_state[zero_pos], new_state[new_pos] = new_state[new_pos], new_state[zero_pos]
            neighbors.append(tuple(new_state))

    return neighbors
```

[ ]:
```python
def construct_path(goal, parents):
    path = []
    state = goal
    while state is not None:
        path.append(state)
        state = parents[state]
    path.reverse()
    return path
```

jupyter iterative_deepening_depth_limited_search Last Checkpoint: 58 minutes ago

File Edit View Run Kernel Settings Help

Code

Ju

[ ]:
```python
def construct_path(goal, parents):
    path = []
    state = goal
    while state is not None:
        path.append(state)
        state = parents[state]
    path.reverse()
    return path
```


[ ]:
```python
def iddfs(start, goal, max_depth=20):
    for depth in range(max_depth + 1):
        visited = set()
        parents = {start: None}
        result = dls(start, goal, depth, visited, parents)
        if result is not None:
            return result
    return None
```

[ ]:
```python
start_state = (1, 2, 3, 4, 5, 6, 7, 8, 0)
goal_state = (1, 2, 3, 4, 5, 6, 7, 0, 8)

solution = iddfs(start_state, goal_state)
if solution:
    print("Solution found!")
    for step in solution:
        print(step)
else:
    print("No solution found.")
```

```
No solution found.
```

jupyter Map_colouring Last Checkpoint: 59 minutes ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[3]:
```python
from collections import deque

def solve_map_coloring(graph, colors):
    def backtrack(assignment):
        if len(assignment) == len(graph):
            return assignment
        var = min((v for v in graph if v not in assignment), key=lambda v: len(domains[v]))
        for value in domains[var]:
            if all(assignment.get(nei) != value for nei in graph[var]):
                assignment[var] = value
                if backtrack(assignment):
                    return assignment
                del assignment[var]
        return None

    domains = {node: set(colors) for node in graph}
    return backtrack({}) or "No solution exists for the given map."

if __name__ == "__main__":
    graph = {
        'wa': {'nt', 'sa'}, 'nt': {'wa', 'sa', 'q'}, 'q': {'nt', 'sa', 'nsw'},
        'nsw': {'q', 'sa', 'v'}, 'v': {'sa', 'nsw'}, 'sa': {'wa', 'nt', 'q', 'nsw', 'v'}, 't': set()
    }
    colors = {'Red', 'Green', 'Blue'}
    print("Solutions:", solve_map_coloring(graph, colors))
```

```
Solutions: {'wa': 'Green', 'nt': 'Red', 'q': 'Green', 'nsw': 'Red', 'v': 'Green', 'sa': 'Blue', 't': 'Green'}
```

[ ]:

jupyter sudoku Last Checkpoint: 59 minutes ago

File Edit View Run Kernel Settings Help

Code Jupyterl

[9]:
```python
def is_valid(grid, row, col, num):
    # Check row, column, and 3x3 subgrid
    if num in grid[row] or num in [grid[i][col] for i in range(9)]:
        return False
    subgrid = [grid[r][c] for r in range(row//3*3, row//3*3+3) for c in range(col//3*3, col//3*3+3)]
    return num not in subgrid

def solve_sudoku(grid):
    for row in range(9):
        for col in range(9):
            if grid[row][col] == 0:
                for num in range(1, 10):
                    if is_valid(grid, row, col, num):
                        grid[row][col] = num
                        if solve_sudoku(grid):
                            return True
                        grid[row][col] = 0
                return False
    return True

def print_sudoku(grid):
    for row in grid:
        print(" ".join(map(str, row)))

# Example Sudoku puzzle (0 represents empty cells)
initial_grid = [
    [0, 0, 3, 0, 2, 0, 6, 0, 0],
    [9, 0, 0, 3, 0, 5, 0, 0, 1],
    [0, 0, 1, 8, 0, 6, 4, 0, 0],
    [0, 0, 8, 1, 0, 2, 9, 0, 0],
    [7, 0, 0, 0, 0, 0, 0, 0, 8],
    [0, 0, 6, 7, 0, 8, 2, 0, 0],
    [0, 0, 2, 6, 0, 9, 5, 0, 0],
    [8, 0, 0, 2, 0, 3, 0, 0, 9],
```

```python
def print_sudoku(grid):
    for row in grid:
        print(" ".join(map(str, row)))

# Example Sudoku puzzle (0 represents empty cells)
initial_grid = [
    [0, 0, 3, 0, 2, 0, 6, 0, 0],
    [9, 0, 0, 3, 0, 5, 0, 0, 1],
    [0, 0, 1, 8, 0, 6, 4, 0, 0],
    [0, 0, 8, 1, 0, 2, 9, 0, 0],
    [7, 0, 0, 0, 0, 0, 0, 0, 8],
    [0, 0, 6, 7, 0, 8, 2, 0, 0],
    [0, 0, 2, 6, 0, 9, 5, 0, 0],
    [8, 0, 0, 2, 0, 3, 0, 0, 9],
    [0, 0, 5, 0, 1, 0, 3, 0, 0]
]

if solve_sudoku(initial_grid):
    print_sudoku(initial_grid)
else:
    print("No solution exists.")
```

```
4 8 3 9 2 1 6 5 7
9 6 7 3 4 5 8 2 1
2 5 1 8 7 6 4 9 3
5 4 8 1 3 2 9 7 6
7 2 9 5 6 4 1 3 8
1 3 6 7 9 8 2 4 5
3 7 2 6 8 9 5 1 4
8 1 4 2 5 3 7 6 9
6 9 5 4 1 7 3 8 2
```

jupyter tic_tac_toe Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Markdown

x wins

[4]:
```python
import math
import random

PLAYER_X, PLAYER_O, EMPTY = 1, 2, 0

def print_board(board):
    symbols = {EMPTY: '.', PLAYER_X: 'X', PLAYER_O: 'O'}
    for row in board:
        print(' '.join(symbols[cell] for cell in row))
    print()

def check_winner(board, player):
    win_patterns = [(0,0,0,1,0,2), (1,0,1,1,1,2), (2,0,2,1,2,2),   # Rows
                    (0,0,1,0,2,0), (0,1,1,1,2,1), (0,2,1,2,2,2),   # Columns
                    (0,0,1,1,2,2), (0,2,1,1,2,0)]                  # Diagonals
    return any(board[x1][y1] == board[x2][y2] == board[x3][y3] == player for x1, y1, x2, y2, x3, y3 in win_patter

def alpha_beta(board, player, alpha, beta):
    if check_winner(board, PLAYER_X): return 10, None
    if check_winner(board, PLAYER_O): return -10, None
    if all(cell != EMPTY for row in board for cell in row): return 0, None  # Draw

    best_move = None
    if player == PLAYER_X:
        max_eval = -math.inf
        for i, row in enumerate(board):
            for j, cell in enumerate(row):
                if cell == EMPTY:
```

jupyter tic_tac_toe Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Markdown

```python
def alpha_beta(board, player, alpha, beta):
    if check_winner(board, PLAYER_X): return 10, None
    if check_winner(board, PLAYER_O): return -10, None
    if all(cell != EMPTY for row in board for cell in row): return 0, None  # Draw

    best_move = None
    if player == PLAYER_X:
        max_eval = -math.inf
        for i, row in enumerate(board):
            for j, cell in enumerate(row):
                if cell == EMPTY:
                    board[i][j] = PLAYER_X
                    eval, _ = alpha_beta(board, PLAYER_O, alpha, beta)
                    board[i][j] = EMPTY
                    if eval > max_eval: max_eval, best_move = eval, (i, j)
                    alpha = max(alpha, eval)
                    if beta <= alpha: break
        return max_eval, best_move
    else:
        min_eval = math.inf
        for i, row in enumerate(board):
            for j, cell in enumerate(row):
                if cell == EMPTY:
                    board[i][j] = PLAYER_O
                    eval, _ = alpha_beta(board, PLAYER_X, alpha, beta)
                    board[i][j] = EMPTY
                    if eval < min_eval: min_eval, best_move = eval, (i, j)
                    beta = min(beta, eval)
                    if beta <= alpha: break
        return min_eval, best_move

def main():
    board = [[EMPTY]*3 for _ in range(3)]
    print("Initial Board:")
```

jupyter tic_tac_toe Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Markdown

```python
def main():
    board = [[EMPTY]*3 for _ in range(3)]
    print("Initial Board:")
    print_board(board)

    while True:
        # Player X's move (AI)
        _, move = alpha_beta(board, PLAYER_X, -math.inf, math.inf)
        if move:
            board[move[0]][move[1]] = PLAYER_X
            print("Player X's move:")
            print_board(board)
        if check_winner(board, PLAYER_X): print("Player X wins!"); break
        if all(cell != EMPTY for row in board for cell in row): print("It's a draw!"); break

        # Player O's move (Random)
        move = random.choice([(i, j) for i in range(3) for j in range(3) if board[i][j] == EMPTY])
        board[move[0]][move[1]] = PLAYER_O
        print("Player O's move:")
        print_board(board)
        if check_winner(board, PLAYER_O): print("Player O wins!"); break
        if all(cell != EMPTY for row in board for cell in row): print("It's a draw!"); break

if __name__ == "__main__":
    main()
```

```
Initial Board:
. . .
. . .
. . .

Player X's move:
X . .
```

jupyter torch Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code

JupyterLab

[1]:

```python
from collections import deque

def solve_bridge_crossing():
    # State format: (flashlight_side, you, wolf, goat, cabbage, total_time)
    start_state = (0, 0, 0, 0, 0, 0)  # Everyone starts on the left side with 0 time spent

    # Goal state: everyone on the right side
    goal_state = (1, 1, 1, 1, 1)

    # BFS queue: stores (current state, path taken)
    queue = deque([(start_state, [start_state])])

    # Set to store visited states
    visited = set([start_state])

    # Crossing times for each character (you, wolf, goat, cabbage)
    crossing_times = [1, 2, 5, 10]

    # BFS loop
    while queue:
        current_state, path = queue.popleft()
        flashlight_side, you, wolf, goat, cabbage, total_time = current_state

        # Check if we have reached the goal
        if (flashlight_side, you, wolf, goat, cabbage) == goal_state:
            return path, total_time

        # Generate possible next states
        next_states = generate_next_states(current_state, crossing_times)

        for next_state in next_states:
            if next_state not in visited:
```

jupyter torch Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code JupyterLab Pyt

```python
        flashlight_side, you, wolf, goat, cabbage, total_time = current_state

        # Check if we have reached the goal
        if (flashlight_side, you, wolf, goat, cabbage) == goal_state:
            return path, total_time

        # Generate possible next states
        next_states = generate_next_states(current_state, crossing_times)

        for next_state in next_states:
            if next_state not in visited:
                visited.add(next_state)
                queue.append((next_state, path + [next_state]))

    # No solution found
    return None, None

def generate_next_states(state, crossing_times):
    flashlight_side, you, wolf, goat, cabbage, total_time = state
    current_positions = [you, wolf, goat, cabbage]
    next_states = []

    # Determine the direction of crossing based on the flashlight's position
    if flashlight_side == 0:  # Moving from left to right
        possible_moves = [(i, j) for i in range(4) for j in range(i, 4) if current_positions[i] == 0 and current_positions[j] == 0]
    else:  # Moving from right to left
        possible_moves = [(i, j) for i in range(4) for j in range(i, 4) if current_positions[i] == 1 and current_positions[j] == 1]

    for i, j in possible_moves:
        # Determine the time taken for this move
        crossing_time = max(crossing_times[i], crossing_times[j])

        # Update positions after crossing
        new_positions = current_positions[:]
        new_positions[i] = 1 - new_positions[i]  # Flip the side
```


jupyter torch Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code JupyterLab Pyt

```python
def generate_next_states(state, crossing_times):
    flashlight_side, you, wolf, goat, cabbage, total_time = state
    current_positions = [you, wolf, goat, cabbage]
    next_states = []

    # Determine the direction of crossing based on the flashlight's position
    if flashlight_side == 0:  # Moving from left to right
        possible_moves = [(i, j) for i in range(4) for j in range(i, 4) if current_positions[i] == 0 and current_positions[j] == 0]
    else:  # Moving from right to left
        possible_moves = [(i, j) for i in range(4) for j in range(i, 4) if current_positions[i] == 1 and current_positions[j] == 1]

    for i, j in possible_moves:
        # Determine the time taken for this move
        crossing_time = max(crossing_times[i], crossing_times[j])

        # Update positions after crossing
        new_positions = current_positions[:]
        new_positions[i] = 1 - new_positions[i]  # Flip the side
```


```python
        new_positions[j] = 1 - new_positions[j]  # Flip the side

        # Update the flashlight's position and total time
        new_flashlight_side = 1 - flashlight_side
        new_total_time = total_time + crossing_time

        # Create the new state
        new_state = (new_flashlight_side, *new_positions, new_total_time)

        # Add to the list of next valid states
        next_states.append(new_state)

    return next_states
```

[2]:
```python
def print_solution(solution, total_time):
    if solution:
        print(f"Solution found in {total_time} minutes:")
        for state in solution:
            flashlight_side = 'Right' if state[0] == 1 else 'Left'
            positions = ['Right' if x == 1 else 'Left' for x in state[1:5]]
            print(f"Flashlight: {flashlight_side}, You: {positions[0]}, Wolf: {positions[1]}, Goat: {positions[2]}, Cabbage: {positions[3]}, Time: {state
    else:
        print("No solution found.")

# Run the solver
solution, total_time = solve_bridge_crossing()
print_solution(solution, total_time)
```

```
Solution found in 13 minutes:
Flashlight: Left, You: Left, Wolf: Left, Goat: Left, Cabbage: Left, Time: 0 minutes
Flashlight: Right, You: Right, Wolf: Right, Goat: Left, Cabbage: Left, Time: 2 minutes
Flashlight: Left, You: Right, Wolf: Right, Goat: Left, Cabbage: Left, Time: 3 minutes
Flashlight: Right, You: Right, Wolf: Right, Goat: Right, Cabbage: Right, Time: 13 minutes
```

[ ]:

[ ]:

jupyter torch Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help Not Trusted

Code JupyterLab Python 3 (ipykernel)

[2]:
```python
ition(solution, total_time):
on:
(f"Solution found in {total_time} minutes:")
tate in solution:
lashlight_side = 'Right' if state[0] == 1 else 'Left'
ositions = ['Right' if x == 1 else 'Left' for x in state[1:5]]
rint(f"Flashlight: {flashlight_side}, You: {positions[0]}, Wolf: {positions[1]}, Goat: {positions[2]}, Cabbage: {positions[3]}, Time: {state[5]} minutes")

("No solution found.")

ver
l_time = solve_bridge_crossing()
n(solution, total_time)
```

```
Solution found in 13 minutes:
Flashlight: Left, You: Left, Wolf: Left, Goat: Left, Cabbage: Left, Time: 0 minutes
Flashlight: Right, You: Right, Wolf: Right, Goat: Left, Cabbage: Left, Time: 2 minutes
Flashlight: Left, You: Right, Wolf: Right, Goat: Left, Cabbage: Left, Time: 3 minutes
Flashlight: Right, You: Right, Wolf: Right, Goat: Right, Cabbage: Right, Time: 13 minutes
```

[ ]:

[ ]:

jupyter TTT_testing Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code JupyterLab Python 3 (i

[1]:
```python
import math
import random

PLAYER_X, PLAYER_O, EMPTY = 1, 2, 0

def print_board(board):
    symbols = {EMPTY: '.', PLAYER_X: 'X', PLAYER_O: 'O'}
    for row in board:
        print(' '.join(symbols[cell] for cell in row))
    print()
```

[2]:
```python
def check_winner(board, player):
    win_patterns = [(0,0,0,1,0,2), (1,0,1,1,1,2), (2,0,2,1,2,2),   # Rows
                    (0,0,1,0,2,0), (0,1,1,1,2,1), (0,2,1,2,2,2),   # Columns
                    (0,0,1,1,2,2), (0,2,1,1,2,0)]                  # Diagonals
    return any(board[x1][y1] == board[x2][y2] == board[x3][y3] == player for x1, y1, x2, y2, x3, y3 in win_patterns)
```

[3]:
```python
def alpha_beta(board, player, alpha, beta):
    if check_winner(board, PLAYER_X): return 10, None
    if check_winner(board, PLAYER_O): return -10, None
    if all(cell != EMPTY for row in board for cell in row): return 0, None  # Draw

    best_move = None
    if player == PLAYER_X:
        max_eval = -math.inf
        for i, row in enumerate(board):
            for j, cell in enumerate(row):
                if cell == EMPTY:
                    board[i][j] = PLAYER_X
                    eval, _ = alpha_beta(board, PLAYER_O, alpha, beta)
                    board[i][j] = EMPTY
                    if eval > max_eval: max_eval, best_move = eval, (i, j)
```

jupyter TTT_testing Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code JupyterLab Pyth

```
                if eval > max_eval: max_eval, best_move = eval, (i, j)
                alpha = max(alpha, eval)
                if beta <= alpha: break
        return max_eval, best_move
    else:
        min_eval = math.inf
        for i, row in enumerate(board):
            for j, cell in enumerate(row):
                if cell == EMPTY:
                    board[i][j] = PLAYER_O
                    eval, _ = alpha_beta(board, PLAYER_X, alpha, beta)
                    board[i][j] = EMPTY
                    if eval < min_eval: min_eval, best_move = eval, (i, j)
                    beta = min(beta, eval)
                    if beta <= alpha: break
        return min_eval, best_move
```

[4]:
```
def main():
    board = [[EMPTY]*3 for _ in range(3)]
    print("Initial Board:")
    print_board(board)

    while True:
        # Player X's move (AI)
        _, move = alpha_beta(board, PLAYER_X, -math.inf, math.inf)
        if move:
            board[move[0]][move[1]] = PLAYER_X
            print("Player X's move:")
            print_board(board)
        if check_winner(board, PLAYER_X): print("Player X wins!"); break
        if all(cell != EMPTY for row in board for cell in row): print("It's a draw!"); break

        # Player O's move (Random)
        move = random.choice([(i, j) for i in range(3) for j in range(3) if board[i][j] == EMPTY])
```

jupyter TTT_testing Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code Jupyter

[4]:
```
def main():
    board = [[EMPTY]*3 for _ in range(3)]
    print("Initial Board:")
    print_board(board)

    while True:
        # Player X's move (AI)
        _, move = alpha_beta(board, PLAYER_X, -math.inf, math.inf)
        if move:
            board[move[0]][move[1]] = PLAYER_X
            print("Player X's move:")
            print_board(board)
        if check_winner(board, PLAYER_X): print("Player X wins!"); break
        if all(cell != EMPTY for row in board for cell in row): print("It's a draw!"); break

        # Player O's move (Random)
        move = random.choice([(i, j) for i in range(3) for j in range(3) if board[i][j] == EMPTY])
        board[move[0]][move[1]] = PLAYER_O
        print("Player O's move:")
        print_board(board)
        if check_winner(board, PLAYER_O): print("Player O wins!"); break
        if all(cell != EMPTY for row in board for cell in row): print("It's a draw!"); break
```

[5]:
```
if __name__ == "__main__":
    main()
```

```
Initial Board:
. . .
. . .
. . .

Player X's move:
X . .
. . .
. . .
```

jupyter TTT_testing Last Checkpoint: 1 hour ago

File Edit View Run Kernel Settings Help

Code JupyterLab Python

[5]:
```python
if __name__ == "__main__":
    main()
```

```
Initial Board:
. . .
. . .
. . .

Player X's move:
X . .
. . .
. . .

Player O's move:
X . .
. . O
. . .

Player X's move:
X . X
. . O
. . .

Player O's move:
X . X
O . O
. . .

Player X's move:
X X X
O . O
. . .

Player X wins!
```

[ ]: