# AI Experiment-3

March 4, 2024

[ ]:
```
class State:
    def __init__(self, missionaries, cannibals, boat_position):
        self.missionaries = missionaries
        self.cannibals = cannibals
        self.boat_position = boat_position

    def is_valid(self):
        if (
            0 <= self.missionaries <= 3
            and 0 <= self.cannibals <= 3
            and 0 <= self.boat_position <= 1
        ):
            if (
                self.missionaries == 0
                or self.missionaries == 3
                or self.missionaries >= self.cannibals
            ):
                return True
        return False

    def is_goal(self):
        return self.missionaries == 0 and self.cannibals == 0 and self.
 ↪boat_position == 0

    def __eq__(self, other):
        return (
            self.missionaries == other.missionaries
            and self.cannibals == other.cannibals
            and self.boat_position == other.boat_position
        )

    def __hash__(self):
        return hash((self.missionaries, self.cannibals, self.boat_position))


def generate_next_states(current_state):
    next_states = []
```

```
    moves = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]

    for m, c in moves:
        if current_state.boat_position == 1:
            new_state = State(
                current_state.missionaries - m,
                current_state.cannibals - c,
                0,
            )
        else:
            new_state = State(
                current_state.missionaries + m,
                current_state.cannibals + c,
                1,
            )

        if new_state.is_valid():
            next_states.append(new_state)

    return next_states


def dfs_search():
    start_state = State(3, 3, 1)
    goal_state = State(0, 0, 0)

    stack = [(start_state, [])]
    visited = set()

    while stack:
        current_state, path = stack.pop()

        if current_state.is_goal():
            return path

        if current_state not in visited:
            visited.add(current_state)

            next_states = generate_next_states(current_state)
            for next_state in next_states:
                if next_state not in visited:
                    stack.append((next_state, path + [current_state]))

    return None


def print_state_description(state):
```

```
    left_shore = f"{state.missionaries} Missionaries and {state.cannibals}␣
↪Cannibals on the Left Shore"
    right_shore = f"{3 - state.missionaries} Missionaries and {3 - state.
↪cannibals} Cannibals on the Right Shore"

    print(f"{left_shore}, {right_shore}\n")


if __name__ == "__main__":
    solution_path = dfs_search()

    if solution_path:
        print("Solution Path:")
        for i, state in enumerate(solution_path):
            print(f"Step {i + 1}:")
            print_state_description(state)

    else:
        print("No solution found.")
```

```
Solution Path:
Step 1:
3 Missionaries and 3 Cannibals on the Left Shore, 0 Missionaries and 0 Cannibals
on the Right Shore

Step 2:
2 Missionaries and 2 Cannibals on the Left Shore, 1 Missionaries and 1 Cannibals
on the Right Shore

Step 3:
3 Missionaries and 2 Cannibals on the Left Shore, 0 Missionaries and 1 Cannibals
on the Right Shore

Step 4:
2 Missionaries and 1 Cannibals on the Left Shore, 1 Missionaries and 2 Cannibals
on the Right Shore

Step 5:
2 Missionaries and 2 Cannibals on the Left Shore, 1 Missionaries and 1 Cannibals
on the Right Shore

Step 6:
1 Missionaries and 1 Cannibals on the Left Shore, 2 Missionaries and 2 Cannibals
on the Right Shore

Step 7:
3 Missionaries and 1 Cannibals on the Left Shore, 0 Missionaries and 2 Cannibals
```

```
on the Right Shore

Step 8:
2 Missionaries and 0 Cannibals on the Left Shore, 1 Missionaries and 3 Cannibals
on the Right Shore

Step 9:
2 Missionaries and 1 Cannibals on the Left Shore, 1 Missionaries and 2 Cannibals
on the Right Shore

Step 10:
1 Missionaries and 0 Cannibals on the Left Shore, 2 Missionaries and 3 Cannibals
on the Right Shore

Step 11:
1 Missionaries and 1 Cannibals on the Left Shore, 2 Missionaries and 2 Cannibals
on the Right Shore
```

[ ]:
```python
def pour_water(state, action):
    x, y = state
    if action == 'fill_4':
        return (4, y)
    elif action == 'fill_3':
        return (x, 3)
    elif action == 'empty_4':
        return (0, y)
    elif action == 'empty_3':
        return (x, 0)
    elif action == 'pour_4_to_3':
        amount = min(x, 3 - y)
        return (x - amount, y + amount)
    elif action == 'pour_3_to_4':
        amount = min(y, 4 - x)
        return (x + amount, y - amount)
    else:
        return state

def dfs(state, visited):
    if state[0] == 2:
        return [state]
    visited.add(state)
    for action in ['fill_4', 'fill_3', 'empty_4', 'empty_3', 'pour_4_to_3',␣
 ↪'pour_3_to_4']:
        new_state = pour_water(state, action)
        if new_state not in visited:
            path = dfs(new_state, visited)
```

```
            if path:
                return [state] + path
    return None

def print_steps(path):
    for i, state in enumerate(path):
        jug_4, jug_3 = state
        print(f"Step {i+1}: Jug 4: {jug_4} gallons, Jug 3: {jug_3} gallons")

initial_state = (0, 0)
visited = set()
path = dfs(initial_state, visited)

if path:
    print("Steps to measure 2 gallons:")
    print_steps(path)
else:
    print("No solution found.")
```

```
Steps to measure 2 gallons:
Step 1: Jug 4: 0 gallons, Jug 3: 0 gallons
Step 2: Jug 4: 4 gallons, Jug 3: 0 gallons
Step 3: Jug 4: 4 gallons, Jug 3: 3 gallons
Step 4: Jug 4: 0 gallons, Jug 3: 3 gallons
Step 5: Jug 4: 3 gallons, Jug 3: 0 gallons
Step 6: Jug 4: 3 gallons, Jug 3: 3 gallons
Step 7: Jug 4: 4 gallons, Jug 3: 2 gallons
Step 8: Jug 4: 0 gallons, Jug 3: 2 gallons
Step 9: Jug 4: 2 gallons, Jug 3: 0 gallons
```