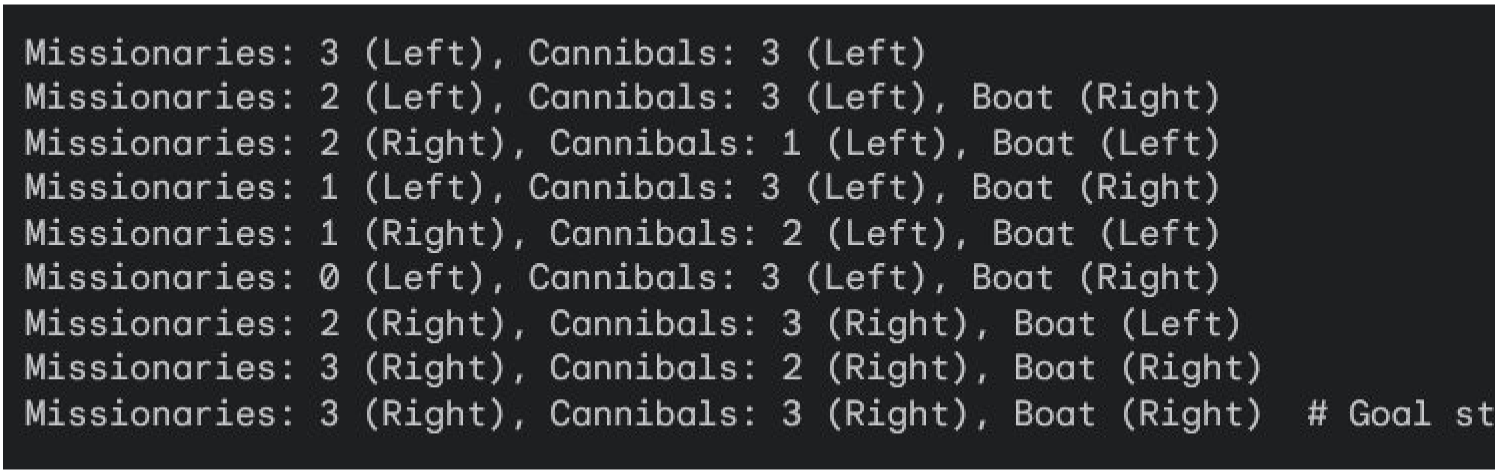
Sumedh ahire
FYMCA-B 03
BATCH 1
ASSIGNEMT 1

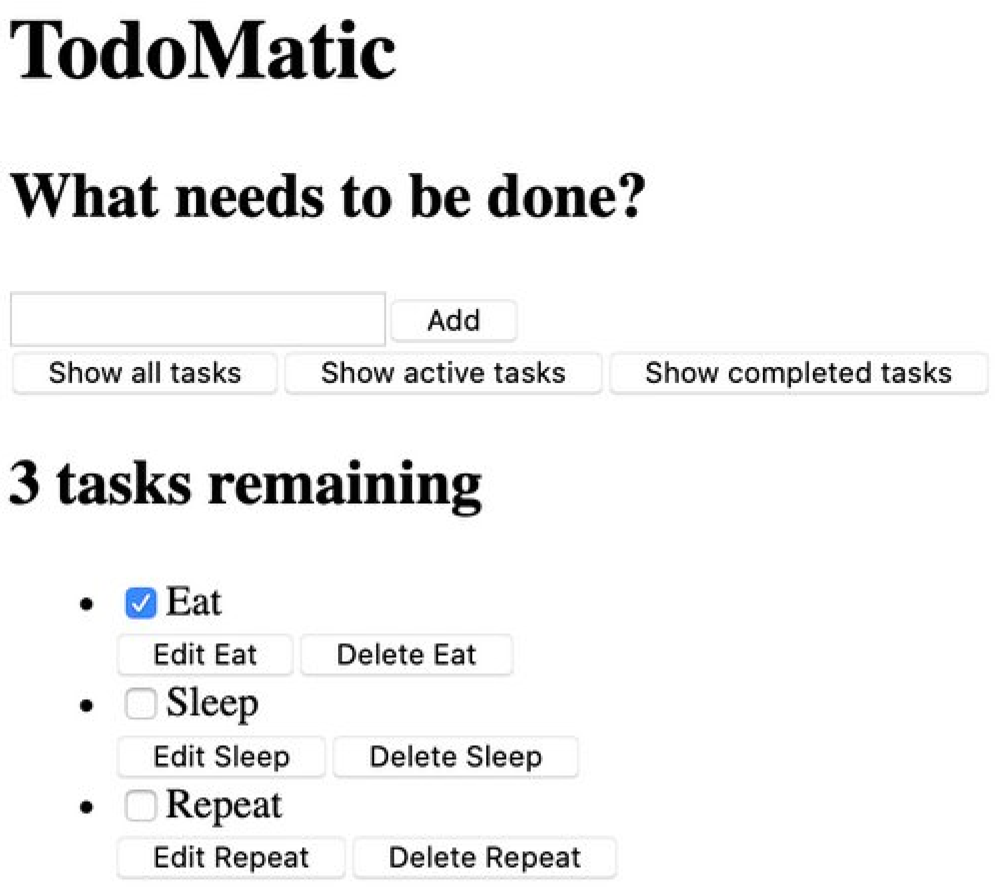
CODE:

```
from collections import deque

class State:
  def __init__(self, missionaries_left, cannibals_left, boat, parent=None):
    self.missionaries_left = missionaries_left
    self.cannibals_left = cannibals_left
    self.boat = boat  # True: boat on left, False: boat on right
    self.parent = parent

  def __hash__(self):
    return hash((self.missionaries_left, self.cannibals_left, self.boat))

  def __eq__(self, other):
    return (self.missionaries_left == other.missionaries_left and
            self.cannibals_left == other.cannibals_left and
            self.boat == other.boat)

  def __str__(self):
    location = "Left" if self.boat else "Right"
    return f"Missionaries: {self.missionaries_left} ({location}), Cannibals: {self.cannibals_left} ({location})"

def is_valid(state):
  # Check if cannibal count exceeds missionary count on either side
  missionaries_left, cannibals_left, _ = state
  if (missionaries_left > 0 and cannibals_left > missionaries_left) or \
     (3 - missionaries_left > 0 and 3 - cannibals_left > 3 - missionaries_left):
    return False
  return True

def generate_successors(state):
  successors = []
  missionaries_left, cannibals_left, boat = state
  # Generate all possible moves (up to 2 people)
  for m in range(min(missionaries_left, 2) + 1):
    for c in range(min(cannibals_left, 2) + 1):
      if m + c > 0:  # Must move at least one person
        new_missionaries_left = missionaries_left - m if boat else missionaries_left + m
        new_cannibals_left = cannibals_left - c if boat else cannibals_left + c
        new_boat = not boat
        if is_valid(State(new_missionaries_left, new_cannibals_left, new_boat, state)):
          successors.append(State(new_missionaries_left, new_cannibals_left, new_boat, state))
  return successors

def heuristic(state):
  # Heuristic: Distance to goal state (all missionaries and cannibals on right side)
  missionaries_left, cannibals_left, _ = state
  return missionaries_left + cannibals_left

def a_star_search(initial_state):
  open_set = deque([initial_state])
  closed_set = set()
  g_score = {initial_state: 0}
  f_score = {initial_state: heuristic(initial_state)}

  while open_set:
    current = min(open_set, key=lambda state: f_score[state])
    open_set.remove(current)
    closed_set.add(current)

    if current.missionaries_left == 0 and current.cannibals_left == 0 and not current.boat:
      # Goal state reached
      path = []
      while current:
        path.append(current)
        current = current.parent
      return path[::-1]  # Reverse path to get order of moves

    for successor in generate_successors(current):
      if successor in closed_set:
        continue

      tentative_g_score = g_score[current] + 1

      if successor not in open_set or tentative_g_score < g_score.get(successor, float('inf')):
        g_score[successor] = tentative_g_score
        f_score[successor] = tentative_g_score + heuristic(successor)
        open_set.append(successor)

  return None  # No solution found

# Initial state: All missionaries and cannibals on the left side, boat on left
initial_state = State(3, 3, True)

solution = a_star_search(initial_state)

if solution:
  for state in solution:
    print(state)
else:
  print("No solution found")
```

OUTPUT:



```
Missionaries: 3 (Left), Cannibals: 3 (Left)
Missionaries: 2 (Left), Cannibals: 3 (Left), Boat (Right)
Missionaries: 2 (Right), Cannibals: 1 (Left), Boat (Left)
Missionaries: 1 (Left), Cannibals: 3 (Left), Boat (Right)
Missionaries: 1 (Right), Cannibals: 2 (Left), Boat (Left)
Missionaries: 0 (Left), Cannibals: 3 (Left), Boat (Right)
Missionaries: 2 (Right), Cannibals: 3 (Right), Boat (Left)
Missionaries: 3 (Right), Cannibals: 2 (Right), Boat (Right)
Missionaries: 3 (Right), Cannibals: 3 (Right), Boat (Right)  # Goal st
```



TodoMatic

What needs to be done?

3 tasks remaining

- ☑ Eat
- □ Sleep
- □ Repeat