Missionaries and cannibals

Github:

https://github.com/pjavier98/AI/tree/master/missionaries_and_cannibals

Archive: missionaries_and_cannibals.py

```
from node import *
from util import *

def generate_solution(queue, visited_states, util):
  initial_state = State(3, 0, 3, 0, 'left')

  visited_states = [initial_state]
  queue = [initial_state]

  for state in queue:
    if state.final_state():
      util.solution = [state]
      while state.dad:
        util.solution.insert(0, state.dad)
        state = state.dad
      break

    state.generate_children(visited_states)
    queue.extend(state.children)

def print_solution(solution):
  index = 1
  print('\n\t\tSolution: Missionaries and Cannibals\n')
  for state in solution:
    print(65 * '#')
    print('#\tState ' + str(index) + ': \t\t\t\t\t\t#\n' + str(state))
    print(65 * '#', end='\n\n')
    index += 1

def main():
  util = Util()
  generate_solution(util.queue, util.visited_states, util)
  print_solution(util.solution)

main()
```

Archive: state.py

```
class State():

  def __init__(self, missionaries_left, missionaries_right, cannibals_left,
cannibals_right, boat_side):
    self.missionaries_left = missionaries_left
    self.missionaries_right = missionaries_right
    self.cannibals_left = cannibals_left
    self.cannibals_right = cannibals_right
    self.boat_side = boat_side
    self.dad = None
    self.children = []

  def __str__(self):
    if (self.boat_side == 'left'):
      return '#\tMissionaries Left: {}\t|\tMissionaries Right:
{}\t#\n#\tCannibals Left: {}\t|\tCannibals Right: {}\t#\n#\tBoat Side:
{}\t\t\t\t\t\t#'.format(
        self.missionaries_left, self.missionaries_right,
self.cannibals_left, self.cannibals_right, self.boat_side
      )
    else:
      return '#\tMissionaries Left: {}\t|\tMissionaries Right:
{}\t#\n#\tCannibals Left: {}\t|\tCannibals Right: {}\t#\n#\tBoat Side:
{}\t\t\t\t\t#'.format(
        self.missionaries_left, self.missionaries_right,
self.cannibals_left, self.cannibals_right, self.boat_side
      )


  def valid_state(self):
    # Cannot generate states where the number of cannibals and missionaries
is negative
    if ((self.missionaries_left < 0) or (self.missionaries_right < 0)
      or (self.cannibals_left < 0) or (self.cannibals_right < 0)):
      return False

    '''
      Verify if is equal to zero because may not have missionaries on one
      of the river banks

      Missionaries Left: 3    |       Missionaries Right: 0
      Cannibals Left: 1       |       Cannibals Right: 2
    '''

    '''
      Verify if in both river banks the amount of missionaries is biggest
      or equal than the amount of cannibals

      Missionaries Left: 2    |       Missionaries Right: 1
      Cannibals Left: 2       |       Cannibals Right: 1
    '''
    return ((self.missionaries_left == 0 or self.missionaries_left >=
self.cannibals_left) and
```

```
            (self.missionaries_right == 0 or self.missionaries_right >=
self.cannibals_right))

  def final_state(self):
    # Is the final state if it's one of the answers to the problem
    # If both missionaries and cannibals crossed the river
    result_left = self.missionaries_left == self.cannibals_left == 0
    result_right = self.missionaries_right == self.cannibals_right == 3
    return result_left and result_right

  def check_exists(self, visited_states):
    for state in visited_states:
      miss_left = self.missionaries_left == state.missionaries_left
      miss_right = self.missionaries_right == state.missionaries_right
      cann_left = self.cannibals_left == state.cannibals_left
      cann_right = self.cannibals_right == state.cannibals_right
      bote_side = self.boat_side == state.boat_side
      if (miss_left and miss_left and cann_left and cann_right and
bote_side):
        return True
    return False

  def generate_children(self, visited_states):
    # Generates all possible children of a state if it is a valid state
    # and not is an end state.

    # Find the new side of the river
    boat_side = 'right' if self.boat_side == 'left' else 'left'

    # Possible moves
    moviments = [
      {'missionaries': 0, 'cannibals': 1},
      {'missionaries': 0, 'cannibals': 2},
      {'missionaries': 1, 'cannibals': 1},
      {'missionaries': 1, 'cannibals': 0},
      {'missionaries': 2, 'cannibals': 0},
    ]

    # Generates the possible future movements
    for move in moviments:
      if self.boat_side == 'left':
          # Boat (Left -> Right) (-Left / +Right)
        missionaries_left = self.missionaries_left - move['missionaries']
        missionaries_right = self.missionaries_right + move['missionaries']
        cannibals_left = self.cannibals_left - move['cannibals']
        cannibals_right = self.cannibals_right + move['cannibals']
      else:
        # Boat (Right -> Left) (+Left / -Right)
        missionaries_left = self.missionaries_left + move['missionaries']
        missionaries_right = self.missionaries_right - move['missionaries']
        cannibals_left = self.cannibals_left + move['cannibals']
        cannibals_right = self.cannibals_right - move['cannibals']

      # Create the children state
```

```
        son = State(missionaries_left, missionaries_right, cannibals_left,
                cannibals_right, boat_side)

        if (son.check_exists(visited_states)) == False:

          # Add the new state in the list of visited states
          visited_states.append(son)
          son.dad = self

          # Check if it is a valid state
          if son.valid_state():
            self.children.append(son)
```

Archive: util.py

```
class Util:

  def __init__(self):
    self.visited_states = []
    self.queue = []
    self.solution = []
```