AUCSC 460 Lab 2

First Name: Nathan Last Name: Pelletier ID: 1840408

1. Goal of Lab 2

1.1 After reading a question, be able to formulate a goal, a problem, states, and actions. Then, be able to design state space and transition model

1.2 Understand the depth limited search algorithm and be able to implement the algorithm using Python

2. Getting Credit for the Lab This lab handout has questions.

2.1 As part of the lab, you are to write the answers to these questions, and implement a Python code.

2.2 Before the class ends, instructor will check your (unfinished) answers to record effort. **Labs are graded on effort and correctness**. The instructor may ask you a few questions to make sure you understand the material and then record your success.

2.3 You then submit two files (lab2.pdf file after answering the questions, and lab2.py) onto the eclass lab assignment

In case that you do not finish it during lab, you can submit the two files on the eclass by due date (**Students who missed lab cannot upload lab document.**):
by 11:59 pm Tuesday next week.
The full marks are 100 pts.
If your submission date is Wednesday, you will have 10 pts deduction.
If your submission date is Thursday, you will have 20 pts deduction.
If your submission date is after Thursday, your submission will not be marked. You will get zero mark.

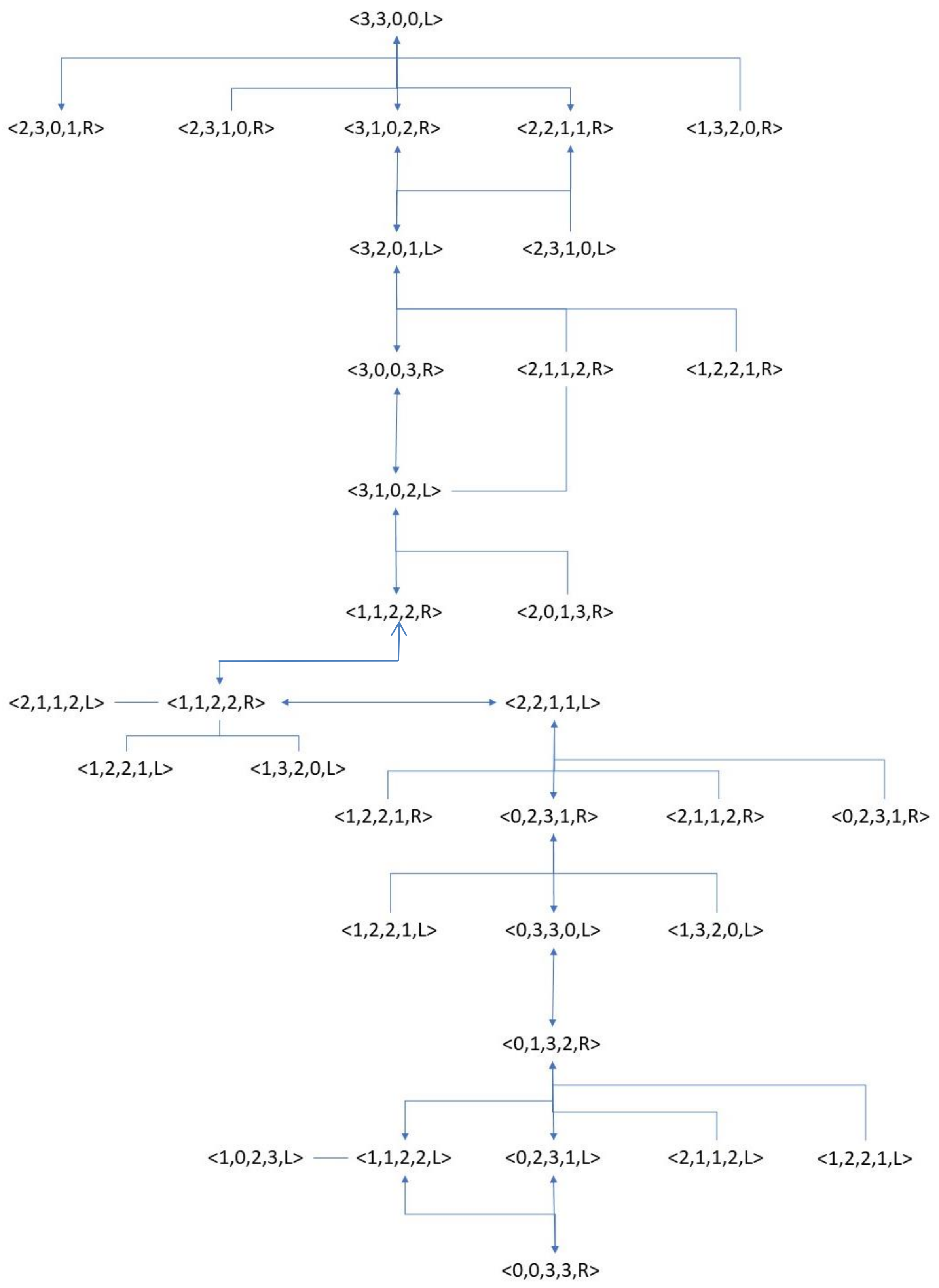
You can discuss all the following problems with classmates, but do not look at others' solutions or copy them.

3. Solve the following problems.

(1) Solve 3.9(a).
* Always the boat holds one or two people.
* Missionaries should not be outnumbered by the cannibals on each side.

Represent a state as a six-tuple of integer listing the number of missionaries, cannibals, and existence of the boat on the first side (sideA), and then the number of missionaries, cannibals, and existence of the boat on the second side (sideB). The start state is (3, 3, 1, 0, 0, 0) which means 3 missionaries, 3 cannibals on the sideA, and boat is also on the sideA. On the sideB, no missionaries, no cannibals and no boat. The goal state is (0,0,0,3,3,1) which means 3 missionaries, 3 cannibals on the sideB and boat is also on the sideB.
Draw a diagram of the complete state space and transition model using edges. (Remove the same states and paths. It is natural that the transition model has cycles.)

(You can insert a photo once you clearly and neatly draw the diagram on an empty paper.)


<3,3,0,0,L>
<2,3,0,1,R>
<2,3,1,0,R>
<3,1,0,2,R>
<2,2,1,1,R>
<1,3,2,0,R>
<3,2,0,1,L>
<2,3,1,0,L>
<3,0,0,3,R>
<2,1,1,2,R>
<1,2,2,1,R>
<3,1,0,2,L>
<1,1,2,2,R>
<2,0,1,3,R>
<2,1,1,2,L>
<1,1,2,2,R>
<2,2,1,1,L>
<1,2,2,1,L>
<1,3,2,0,L>
<1,2,2,1,R>
<0,2,3,1,R>
<2,1,1,2,R>
<0,2,3,1,R>
<1,2,2,1,L>
<0,3,3,0,L>
<1,3,2,0,L>
<0,1,3,2,R>
<1,0,2,3,L>
<1,1,2,2,L>
<0,2,3,1,L>
<2,1,1,2,L>
<1,2,2,1,L>
<0,0,3,3,R>

(2) Solve 3.9(b) using Depth-limited search with limit 15.
The algorithm is in Figure 3.17. Your code should follow the process of Figure 3.17. (This problem is to check if students understand the pseudo-code algorithm of Figure 3.17 and can implement the algorithm using Python.) You can do minor changes of the algorithm of Figure 3.17, but no major changes. Although you already figured out what kinds of children nodes (states) are possible for each node in the question (1), your program does not know the information in advance. For each node, there are five possible actions (please think why five.), and your program will compute and figure out each possible child node (state) at each time.

There are functions *DEPTH-LIMITED-SEARCH(problem, limit), and RECURSIVE-DLS(node, problem, limit)* in Figure 3.17. Because our target problem is only one, you can remove 'problem' from the arguments. You have to explain each action that was chosen using the words missionary, cannibal, side A and side B (e.g. MOVE 1 missionary and 1 cannibal from sideA to sideB), and then also describe which state you arrived in after taking each action. (e.g., the next state is (2,2,0,1,1,1)

(a) You can follow the algorithm below for this lab.

```
function DEPTH-LIMITED-SEARCH( limit )  returns a solution, or failure/cutoff
    return RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE), none, limit )

function RECURSIVE-DLS(node, action, limit )  returns a solution, or failure/cutoff
  if GOAL-TEST(node.STATE)  then return action
  else if limit = 0  then return cutoff
  else
      cutoff_occurred?←false
      for each action  in ACTIONS(node.STATE)  do
          child ←CHILD-NODE(node, action)
          result_action ←RECURSIVE-DLS(child , limit − 1)
          if result = cutoff  then cutoff_occurred?←true
          else if result ≠failure
                   then print("At the current ", node,  " you will take the action ", action
                        (explain action in English) , " and then, your new state is " , child)
                   return result
      if cutoff_occurred?  then return cutoff  else return failure
```

(b) Your program's output should be like this (Because this algorithm is recursive, the right order of the action sequence to arrive in the goal is from the bottom to the top.)

```
At the current  (1, 1, 1, 2, 2, 0)  you take the action  <MOVE 1 missionary and 1 cannibal from
  sideA to sideB>  , and then your new state is  (0, 0, 0, 3, 3, 1) (GOAL)
….
….
…
…
..
….
…
…
…..
At the current  (3, 3, 1, 0, 0, 0)  you take the action  <MOVE 1 cannibal from sideA to
  sideB>  , and then your new state is  (3, 2, 0, 0, 1, 1)
```

Your transition model and algorithm will allow many cycles. But you don't need to worry about it, because we put limitation of the tree search depth. Eventually your algorithm will find a solution for you.

My programs results:

```
>>> start()
Goal state achieved
( 0 , 0 , 0 , 3 , 3 , 1)
2  remaining turns
( 1 , 1 , 1 , 2 , 2 , 0)
3  remaining turns
( 0 , 1 , 0 , 3 , 2 , 1)
4  remaining turns
( 0 , 3 , 1 , 3 , 0 , 0)
5  remaining turns
( 0 , 2 , 0 , 3 , 1 , 1)
6  remaining turns
( 2 , 2 , 1 , 1 , 1 , 0)
7  remaining turns
( 1 , 1 , 0 , 2 , 2 , 1)
8  remaining turns
( 3 , 1 , 1 , 0 , 2 , 0)
9  remaining turns
( 3 , 0 , 0 , 0 , 3 , 1)
10  remaining turns
( 3 , 2 , 1 , 0 , 1 , 0)
11  remaining turns
( 3 , 1 , 0 , 0 , 2 , 1)
12  remaining turns
( 3 , 3 , 1 , 0 , 0 , 0)
```

```
13  remaining turns
( 3 , 2 , 0 , 0 , 1 , 1)
14  remaining turns
( 3 , 3 , 1 , 0 , 0 , 0)
15  remaining turns
```

(3) Is your solution optimal? Explain your answer.

No, my program can loop back on itself, it can trap itself in larger loops and it can take 15 moves to find its goal. The optimal goal is 13 moves so with a tighter timer it is optimal.