Artificial Intelligence Assignment

BLOCKS WORLD PROBLEM

STUDENT: POP DIANA-ŞTEFANIA

GROUP: C.EN. 2.2A

YEAR: II

Technical report

1 Problem statement

A set of wooden blocks of various shapes and colors are sitting on a table. The goal is to build one or more vertical stacks of blocks. The catch is that only one block may be moved at a time: it may either be placed on the table or placed atop another block. Because of this, any blocks that are, at a given time, under another block cannot be moved.
Initial state : Given configuration of blocks and a set of block stacks.
Actions and transitions : Move block from the top of one stack onto the table or onto the top of another stack.
Goal : A given final configuration of the stacks of blocks.

2 Pseudocode

Presented below are the algorithms used for defining the heuristics, for computing the resulted state after performing a certain action and for determining the possible actions:

ACTIONS (*state*)
▷ Computes the possible actions to be taken in the current state and returns their list

```
for stack in state do
    for other_stack in state do
        if other_stack != stack then
            append (actions_list, (state → index(stack),
                                   state → index(other_stack))
    if length(stack) != 1 then
        append (actions_list, (state → index(stack), none)
return actions_list
```

Note : none (no stack) can be represented as anything
(for example ' '), except as a natural number (used to enumerate the stack indexes).

RESULT (*state*, *action*)

▷ Computes the resulting state based on a certain action and the current state

```
source_stack ← action[0]
destination_stack ← action[1]
moved_block ← last element in state_list[source_stack]
if length(state[source_stack]) != 1 then
    for iterator ← 0, length(state[source_stack] − 1 do
        append(new_stack, state[source_stack][iterator])
    append(state_list, new_stack)
if destination_stack != none then
    remove(state_list, state[destination_stack])
    append(state_list, state[destination_stack] + (moved_block))
else
    append(state_list, (moved_block))
remove(state_list, state[source_stack])
state_list → sort_by(length(stack))
return state_list
```

Note : none (no stack) can be represented as anything (for example ' '), except as a natural number (used to enumerate the stack indexes).

H1 (*node*)

▷ Checks whether a block is in the right place and returns the number of blocks out of place

```
sum ← 0
for stack in node → state do
    for block in stack do
        for other_stack in goal do
            if block in other_stack then
                block_position ← stack → index(block)
                other_position ← other_stack → index(block)
                if block_position == 0 or other_position == 0
                and block_position != other_position
                or stack[block_position-1] != stack[other_position-1] then
                    sum ← sum + 1
                    break
return sum
```

H2 (*node*)

```
▷ Counts the number of moves than need to be done in order
for every block to reach its correct place
sum ← 0
for stack in node → state do
    for other_stack in goal do
        if stack[0] in goal then
            goal_stack ← other_stack
            break
    for block in stack do
        block_position ← stack → index(block)
        if block in goal_stack then
            if block_position == goal_stack → index(block) then
                continue
        sum ← sum + length(stack) − block_position
        for iterator ← block_position, length(stack) do
            stack_block ← stack[iterator]
            stack_position ← stack → index(stack_block)
            if stack_position != 0 then
                for other_stack in goal do
                    if stack_block in other_stack then
                        other_position ← other_stack → index(stack_block)
                        if other_position != 0 then
                            for iterator_2 ← 0, stack_position do
                                other_block ← stack[iterator_2]
                                if other_block in other_stack then
                                    if other_stack → index(other_block)
                                    < other_position then
                                        sum ← sum + 1
                                        break
return sum
```

3 Application design

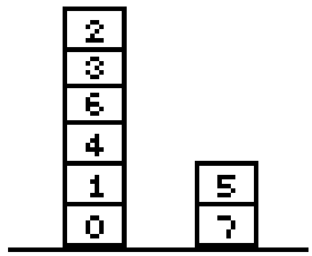
3.1 Architectural overview :

The application is structured in 5 modules : **main.py**, **problem.py**, **blocks_world.py**, **random_state_generator.py**, and **search.py**.

The **problem.py** module contains the class **BlocksWorld** used to define the Blocks World problem, whereas **blocks_world.py** is used to extend the class and implement problem heuristics.

The module **random_state_generator.py** is used to generate a random state according to a number of blocks given.

Figure 1: A random configuration for 8 blocks



A state is represented as a *"tuple of tuples"*, where each tuple contained by the main tuple represents a stack of blocks and their first element represents the base block.

Example: The state from *Figure 1* would be represented as
state = ((0, 1, 4 , 6 , 3, 2), (7, 5))

The module **search.py** is used for implementing the search algorithms (A* and recursive best-first search) and it closely follows the **AIMA framework.**

3.2 Input specification

The application will run using the main.py module, and will ask the user to input a natural (non-zero) number which represents the number of blocks for the blocks world problem. Then, there will be generated a random n-blocks world problem for which both searchers (A* and recursive best-first search) will be tested.

3.3 Output specification

There were made 10 tests using non-trivial input, the results being written in the files **output1.txt - output10.txt.**
The output presents a randomly generated initial and goal state for which the searchers are tested, and both solutions to the problem found by the searchers. A solution represents the solution path (sequence of states) of the problem with the actions that lead to their specific states.

Example : Action (1, 2) - move top block from stack 1 to stack 2

References

[1] https://en.wikipedia.org/wiki/Blocks_world *Blocks World problem.*

[2] https://docs.python.org/3/library/random.html
Random numbers generator.

[3] http://www.d.umn.edu/~kvanhorn/cs2511/discussions/heuristics.html
https://www.d.umn.edu/~gshute/cs2511/projects/Java/assignment6/blocks/blocks.xhtml *Blocks World heuristics*

[4] https://github.com/aimacode/aima-python
AIMA problem framework python.

[5] LaTeXproject site, http://latex-project.org/