University of Hertfordshire
School of Computer Science

Daniel Polani
Hatfield, February 8, 2023

Theory and Practice of Artificial Intelligence • 2022-2023 Semester B

Exercise Sheet 2

Motto: Puzzles and Search

Assignment 2.1

Create a variation of the optimal nim player to address variations of the game rules:

- `nim_2` or `nim_4`: a different number of sticks to be taken (up to 2, or up to 4)
- `nim_last_loses`: a different winning condition (whoever takes the last stick, *loses*)!

Assignment 2.2

Modify the Hero/Sidekick Problem in terms of state variables, start node, goal node and transitions to handle 2 heros with 2 sidekicks only.

Assignment 2.3*

Formulate the 6-Coins Problem in terms of state variables, start node, goal node and transitions and write a class that represents it.

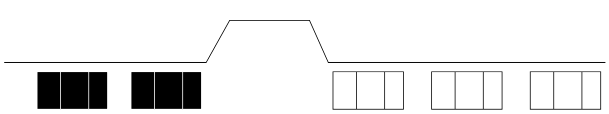
```
import search
import path
import best_first_search

class Six_Coins(search.Nodes):
    def cleanup(self, node):
        # in-place!!

        # remove dummy entries from begin and end of list
        # clean from the front
        while not node[0]:
            node.pop(0)
        while not node[-1]:
            node.pop()

    def start(self):
        return [1,2,1,2,1,2]

    def goal(self, node):
        return node == [1,1,1,2,2,2] or node == [2,2,2,1,1,1]

    def succ(self, node):
        for i in range(len(node)-1): # stop at second last
            # always move two adjacent coins to the right
            if not node[i] or not node[i+1]:
```

```
                # if one of them empty, try other move
                continue

            # try all moves
            for target in range(i+1, len(node)+1):
                #print "move from", i, "to", target

                new_node = node[:]       # copy
                doublet = new_node[i:i+2]
                new_node[i:i+2] = [0,0]  # empty them
                new_node.extend([0,0])   # buffer at the end
                if new_node[target:target+2] == [0,0]:
                    # target area empty
                    new_node[target:target+2] = doublet
                    self.cleanup(new_node) # in-place!!
                    if new_node == node:
                        continue
                    #print "Successor:", node, new_node
                    yield new_node


class Six_Coins_Path(path.Path):
    def __le__(self, path2):
        # compare not only length of path,
        # but length of representations
        if self.length < path2.length:
            return True
        # lexicographic
        if self.length == path2.length:
            return max(len(board) for board in self.path) <= max(len(board) for
                 board in path2.path)
        return False

    def __repr__(self):
        return "-".join("".join(str(coin) for coin in state)
                       for state in self.path)

six_coins = Six_Coins()
start_path = Six_Coins_Path(path = [six_coins.start()], length = 1)
print best_first_search.best_first_search(six_coins, [start_path])
```

Listing 1: Solutions/six_coins.py

Assignment 2.4**

Consider the setting as in the following figure:



Assumptions:

1. The road is only wide enough for one car.

2. The black cars want to drive to the right.
3. The white cars want to drive to the left.
4. The niche is one car wide and long and can thus only be occupied by one car.

Question: how to arrange the car movements (driving left/right, entering niche or not) so that the cars can pass each other?

Can you find the shortest such arrangement?

Hint: Plan your state space model carefully. What are the states you wish to distinguish? What are the moves you want to explicitly model?