P4 Intro

Mini Minecraft Planner

Mini Minecraft Planner

Given ingredients and a recipe → output crafted item

Planner to write:

- Is it possible to create item with given ingredients within given time?
- If so, what is the path, i.e. what do you craft first, what do you craft next?

Files in this assignment

These files are provided for you. You should not edit these files, except for (optionally) the "Goals" in crafting.json for running.

- crafting.json
  - Contains available item types
  - Contains crafting recipes
- pyhop.py
  - HTN engine; we edited a few lines for 146 to enable adding ways to prune search branches using heuristics
- travel.py
  - An example that showcases how pyhop works, that comes with pyhop (it still works with our small modification to pyhop)

```json
1  {
2      "Items": [
3          "cart",
4          "coal",
5          "cobble",
6          "ingot",
7          "ore",
8          "plank",
9          "rail",
10         "stick",
11         "wood"
12     ],
13     "Tools": [
14         "bench",
15         "furnace",
16         "iron_axe",
17         "iron_pickaxe",
18         "stone_axe",
19         "stone_pickaxe",
20         "wooden_axe",
21         "wooden_pickaxe"
22     ],
23     "Initial": {
24     },
25     "Goal": {
26         "wooden_axe": 1
27     },
28     "Recipes": {
29         "iron_axe for wood": {
30             "Produces": {
31                 "wood": 1
32             },
33             "Requires": {
34                 "iron_axe": 1
35             },
36             "Time": 1
37         },
38         "punch for wood": {
39             "Produces": {
40                 "wood": 1
41             },
42             "Time": 4
43         },
```

Files in this assignment

These files are frameworks for you to fill out with your code.

- manualHTN.py
  - To ease you into using the pyhop HTN planner, understanding methods and operators
  - Methods can be understood as possible ways of subtasking a task
  - Operators can be understood as actual actions

```
"Recipes": {
 "punch for wood": {
  "Produces": {
   "wood": 1
  },
  "Time": 4
 },...
```

```python
def op_punch_for_wood (state, ID):
        if state.time[ID] >= 4:
                state.wood[ID] += 1
                state.time[ID] -= 4
                return state
        return False

pyhop.declare_operators (op_punch_for_wood)

def produce (state, ID, item):
        if item == 'wood':
                return [('produce_wood', ID)]

def punch_for_wood (state, ID):
        return [('op_punch_for_wood', ID)]

pyhop.declare_methods ('produce_wood', punch_for_wood)

def check_enough (state, ID, item, num):
        if getattr(state,item)[ID] >= num: return []
        return False

def produce_enough (state, ID, item, num):
        return [('produce', ID, item), ('have_enough', ID, item, num)]

pyhop.declare_methods ('have_enough', check_enough, produce_enough)
pyhop.declare_methods ('produce', produce)

pyhop.pyhop(state, [('have_enough', 'agent', 'wood', 1)], verbose=3)
```

Files in this assignment

These files are frameworks for you to fill out with your code.

- manualHTN.py
    - To ease you into using the pyhop HTN planner, understanding methods and operators
    - Methods can be understood as possible ways of subtasking a task

```
# declare state
state = pyhop.State('state')
state.wood = {'agent': 0}
state.time = {'agent': 4}
```

```
 Produces : {
   "wood": 1
 },
 "Time": 4
},...
```

```python
def op_punch_for_wood (state, ID):
    if state.time[ID] >= 4:
        state.wood[ID] += 1
        state.time[ID] -= 4
        return state
    return False

pyhop.declare_operators (op_punch_for_wood)

def produce (state, ID, item):
    if item == 'wood':
        return [('produce_wood', ID)]

def punch_for_wood (state, ID):
    return [('op_punch_for_wood', ID)]

pyhop.declare_methods ('produce_wood', punch_for_wood)

def check_enough (state, ID, item, num):
    if getattr(state,item)[ID] >= num: return []
    return False

def produce_enough (state, ID, item, num):
    return [('produce', ID, item), ('have_enough', ID, item, num)]

pyhop.declare_methods ('have_enough', check_enough, produce_enough)
pyhop.declare_methods ('produce', produce)

pyhop.pyhop(state, [('have_enough', 'agent', 'wood', 1)], verbose=3)
```

Files in this assignment

These files are frameworks for you to fill out with your code.

- autoHTN.py
  - Methods and operators need to be defined programmatically
  - order methods for crafting the same item in a way that preferred methods come first in declare_methods (smart ordering is necessary for all but the simplest cases)
  - More difficult problem cases in assignment cannot be solved in reasonable time without pruning using add_heuristic

```python
def make_method (name, rule):
        def method (state, ID):
                # your code here
                pass
        return method

def declare_methods (data):
        # your code here
        pass

def make_operator (rule):
        def operator (state, ID):
                # your code here
                pass
        return operator

def declare_operators (data):
        # your code here
        pass

###########################################

def add_heuristic (data, ID):
        def heuristic (state, curr_task, tasks, plan, depth,
calling_stack):
                # your code here
                return False # if True, prune this branch

        pyhop.add_check(heuristic)
```

Note...

- The state (a dict), list of tasks (a list), and the currently-accrued partial plan (a list) are passed up and down the search tree as essentially immutables because a new copy is passed each time
- You can assume we only need one of each tool, and explicitly program this in heuristics; other heuristics should be general and not specific to recipes --- for example, nothing of the sort like: "if already have 16 wood, do not make more wood"
- If your code takes longer than a few seconds to run, it is probably taking too long: check that your operators and methods are set up correctly, check that the ordering used in declare_methods is efficient, and check your heuristic

Submit

- manualHTN.py (2 pt)
- autoHTN.py (7 pt)
- A README file that describes the heuristics you chose/programmed (1 pt)
- (Optional Extra Credit) *custom_case.txt* that states the most complex problem your HTN planner can solve in 30 seconds of real-world time (1 pt)

Questions