Tutorial 7
Complex Actions

Up until now, the action tutorials have focused on creating behaviors that employ one action within their execution step. However, PARs system can multiple, or complex, actions. This means that we can chain together action sequences within a given behavior. Lets examine this by creating a carry behavior. This behavior, like the other tutorial behaviors, is included with the OpenPAR system.

Creating the Action

To demonstrate complex behaviors, lets create a carry action. This actions purpose is to pick up an object and move that object to another location. We already have two behaviors that we can chain together (PickUp and Walk). To create a complex action, we simply need to chain these actions together in the execution phase. To do this, create (or examine) a CarryExec.py, and input the following code:

```
def execution_steps(self, agent, obj1,obj2):
        actions = {'COMPLEX':(SEQUENCE,
                              ("PickUp",{'agents':agent,'objects':obj1}),
                              ("Walk",{'agents':agent,'objects':obj2}))};
        return actions
```

Up until now weve been creating Primitive actions, which only contain one action. Complex actions require a bit more information. First, we must say how the actions should execute. SEQUENCE, explains to the system that the first action should finish before the next one begins. Complex actions can also be in parallel, so that both actions execute simultaneously. There are actually three types of parallel actions, and each of them will be examined later. After the type of action is defined, a tuple of each sub-action is sent. This tuple can be arbitrarily long, as PAR will parse each action as its own network.

With complex actions, the conditions for all actions in the set must be met. This means that pre, applicability, and culmination conditions must be met for the action to succeed. This is a blessing, as our applicability and pre-conditions can simply be:

```
def applicability_condition(self, agent, obj1,obj2):
        return 1

def preparatory_spec(self, agent, obj1,obj2):
        return 1
```

and the culmination condition is simply:

```
def culmination_condition(self, agent, obj1,obj2):
        if finishedAction(self.id):
                return 1
        else:
                return 0
```

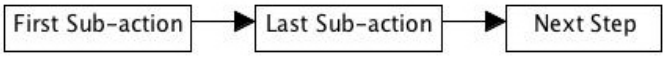
Remember to add this action (and any sub-actions) to the database. Luckily, all of these actions have been added for you.

Inside the code, complex actions work just like primitive actions. They are created and managed like any other iPAR. These actions can be added to the action table as well, and the core-execution of a complex action occurs after all the sub-actions are completed.

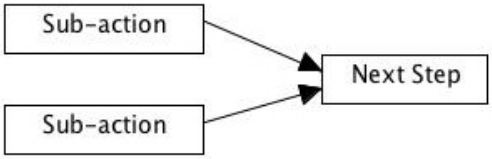
Sequential, Parallel Individual, Parallel Joint, While

The four kinds of complex actions supported as PAR behaviors are sequential (SEQUENCE), parallel independent (PARINDY), parallel joint (PARJOIN), and do while behavior (WHILE).

Sequential actions are the easiest action to understand, as one action is generated after the other.



Parallel independent actions run actions sequentially, and begin the next step once one action finishes. All actions will finish, but that is not a requirement to start the next step. This form of action is most similar to an OR statement.



Parallel joint actions wait for all actions to finish before moving on to the next step. Therefore, if one action is slow, the agent would be focused on that one action and complete that action before moving on.

While actions keep all action states running at the same rate. They run through each sub-actions pre-condition before it runs the execution for any of them. This differs from parallel joint actions in that the actions in parallel joint are not attached at any other node then the last one.