89-674: Introduction to Intelligent, Knowledge-Based, and Cognitive Systems

Spring 2018: EX3

Due at 23:59:59, Tuesday, May 8th, 2018.

Assignment

This assignment has two parts:

Part 1: Implement an executive that uses hierarchal behaviors

In this part you need to implement an executive that executes a version of hierarchical behavior selection shown in the lectures. The hierarchal behaviors in this exercise is such that all higher level behaviors are abstract (they do not have actions of their own), and only the lower level behaviors (leafs of the plan) are action executed in the world.

As a basis, pseudo-code for such executive (BIS) is described in the bis.pdf included in the zip file. Note that BIS algorithm refers to hierarchal plans where higher level behaviors can be executable, you need to change the algorithm so only the lower level behaviors are executed (since the higher level ones are not executable).

For this part we will use simple_football_domain.pddl and simple_football_problem.pddl (included in the zip file), an extremely simplified and discretized version of a soccer-playing agent playing by itself against an empty goal. The field is composed of 14 discrete cells, arranged so:

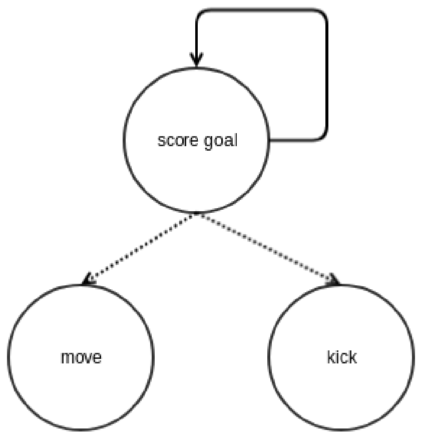
----------------	G0 (R)	G1	G2 (b2)	g3	----------------
start-tile	C0	C1	C2	c3	goal-tile
----------------	D0	D1 (b1)	D2	D3	----------------

The player may be positioned anywhere (here, shown in G0). There are balls (1 or more) positioned also anywhere (here in D1, G2). There are two discrete actions (move and kick). The kick action moves a ball but not the robot. The move action moves the robot but not the ball. All kicks and moves are to the four adjacent cells (north, west, east, south).

The goal for this plan is to get N ball(s) to the goal, where N is given on the command line. It will be an integer (1 or larger).

The problem pddl defines the initial state, which defines where the ball(s) are, and where the robot is. The initial position of the robot and the balls will be changed in our testing. Note that since the PDDL version we use, and the planners we use, cannot count and cannot do loops, it is difficult if not impossible to specify a predicate for the goal condition. Thus instead, true to the ideas of behavior-based control, the PDDL specifies a dummy goal predicate, which will never be true. It is up to the agent to maintain its own beliefs as to how many balls there are, where they are, how many balls have been kicked to the goal, etc. Once the goal is achieved (or your agent believes it is), the agent's next action need to return None, this will stop the simulator. **The report card section at the end of the simulation will tell you that your agent failed: doesn't matter. We will check if the state you achieved is has we expected, N balls in the goal tile.**

To meet this goal, you will write an executive that executes the following hierarchical behavior:



To make this easier, we are providing you with the hierarchical behavior in an XML file (*simple_football_plan.xml*). There is also a python class (*planParser*) which parses the XML file and creates the graph in memory. You need to write an executive that will go over the hierarchical behaviors in the graph and executes them. You will also need to keep track of the number of balls in the goal, etc.

An example usage of planParser:

```
plan_path = "simple_football_plan.xml"
```

```
plan = PlanParser(plan_path).getPlan()
```

To run this part I will run the following command:

```
python football.py -s <N> <problem_file>
```

-s is a flag that tells you to use the simple_football domain and the provided xml (plan_path = "simple_football_plan.xml")

<problem file> will be a PDDL file which defines a testing problem in the simple_football_domain.pddl.

You do not need to modify in any way the simple_football_domain.pddl file, and the simple_football_plan.xml file. You do not need to submit them.

You may (if you wish--optional) improve the parsing python code and include it in your submission. If you do MAKE SURE to explain the improvements in the README file accompanying your submission. Useful improvements will receive bonus points; others will be ignored.

Part 2: Write an hierarchal plan (hierarchical behaviors) of your own

In this part you will need to write an hierarchal plan of your own. This time you will use the extended_football_problem.pddl and extended_football_domain.pddl. The field is the same, and the general objective (N balls in goal) is the same.

However, in this domain you have move, lift and kick action for each leg, left and right. Your agent can move, but it needs to alternate left and right legs. Your agent can kick, but it has to choose the leg with which it is kicking. It cannot kick with the leg it just used for moving (it just finished landing on it---the other one is ready to be lifted for the kick). Once it has kicked with the leg, it can move again with the other leg. The simulated body and domain will not keep track of these requirements: It is your agent’s job to do this, just as it keeps track of the goals it has scored.

The actions are: move-w-left, move-w-right, lift-left, lift-right, kick-left, kick-right.

Example sequences:

- Valid: move-left, move-right, lift-left, kick-left, move right
- Invalid (two left moves): move-left, move-left, move-right
- Invalid (kick without lift): move-right, kick-left
- Invalid (kicking with same leg): move-left, lift-right, lift-right, kick-right
- Invalid (lifting without kicking): move-left, lift-right, move-left

Your code should be able to receive a different problem file and run the plan on it with the preset domain file (domain_path = “extended_football_domain.pddl”) and the plan xml you created(plan_path = "extended_football_plan.xml") and run your plan, **with the same executive from the first part**, on the problem. You **do not** need to submit the domain file or problem file.

You **ONLY** need to submit extended_football_plan.xml.

To run this part I will run the following command:

```
python football.py -e <N> <problem_file>
```

-e is a flag that tells you to use the extended_football domain and your plan xml.

[<N>, <problem_file> same as above]

HINT: You may want to use the hierarchy from part 1 as the basis. Think about extending the hierarchy with more levels.

Part 3. BONUS Part.

The following are optional tasks, with bonus points:

There is actually a way to use a standard PDDL planner to make sure that the behavior-based agent (part 1, part 2) takes the shortest route to each ball, rather than search for it.

- Bonus task 3.1: Describe, in detail, how this can be done for the simple_football_domain. What would be required, how it would work in the executive, and what steps are necessary (5 points bonus).
- Bonus task 3.2: Implement 3.1 and submit as an extended executive. It is OK if it works only for simple_football_domain. (10 points bonus, plus the 5 from 3.1 if done)

The Rules

1. Your code will be run on an ubuntu16 make sure your code works on a this O.S. If your code works on your O.S that isn’t ubuntu16 but won’t work on our computer you will not be able to appeal on your grade.

2. Your work must be performed individually. No group exercises.

What to Hand In

You need to hand in at least 4 files.

- football.py - Your code for both parts.
- extended_football_plan.xml - The plan for part 2.
- README – Contains your name and your ID number and any explanation on your code or plan that you wish to give.
- Optional bonus: planner.docx (or planner.odt) answering 3.1
- Optional bonus: planner_executive.py answering 3.2
- Optional bonus: planParser.py which improves the parsing.

Do not hand in any pddl files.

Finally, zip all the necessary files and submit in https://submit.cs.biu.ac.il/cgi-bin/welcome.cgi .

Good luck!