Project 3: Multi-agent Robots
Computer Science 4013: Artificial Intelligence
Due 12:01AM May 4, 2010
Note: NOT the beginning of class

Introduction

So far, you've made an agent that moves around the world intelligently (project 1) and that can use learning to improve its behavior (project 2). This project focuses on multi-agent systems and teamwork. In this project, you and your partner(s) will control a team of robots. The goal of this project is to intelligently coordinate the (possibly heterogeneous) robots in your team so that you team reaches the flag before any of the other teams.

There are two robots per team and all teams and robots still start at flag 0. In the beginning of the game, the robots are all homogeneous in that no robot has any capability that another robot does not. However, as soon as some robots begin to take damage or to obtain option cards, their abilities diverge. Your goal is to coordinate this diverse team to get to the flag quickly and to cause interference for the other agents. In this project, team performance will be scored by the average number of flags obtained per game.

This project uses planning and logical knowledge representations for the multi-agent coordination but it also integrates aspects of your project 1 and project 2 agents. Your full task is outlined below and we will work on aspects of this in class. All of the projects will use a centralized control approach, where you have a central coach agent that can direct your individual team agents.

- First, identify a set of high-level roles or behaviors appropriate for the individual robots. For each high-level behavior, specify a set of pre-conditions and post-conditions, appropriate for use in planning. These can be specified in ADL or STRIPS.

- Implement a central command agent that determines the roles each agent in your team should take on by using planning, in a ADL or STRIPS-style planner. Ensure that your central command agent replans when events warrant, such as a high-level action completing or sudden loss in energy.

- Implement the set of high-level behaviors as specified by your group. You are not constrained by a specific method for implementing your agent but you should use

what you have learned so far this semester. Heuristics, search, and learning are all appropriate tools in your toolbox.

Extra-credit opportunities

To keep grades within a fair range, all extra credit will be capped at 10 points. This means that no project can receive a grade higher than 110, no matter if they win the ladder, find great exploits, and are very creative.

Wanted dead or alive: bugs or exploits in the simulator

We are reasonably certain that you cannot exploit the simulator (say by directly affecting your opponents damage levels or by directly moving yourself to the goal location). However, any project has bugs and we want to know about them! If you find a bug or an exploit, you can receive extra credit according to the following scale:

- **3 points:** If you find an exploit and report it to us you can receive 3 points extra credit upon verification of the report. Note, we know of two exploits for which you cannot get extra credit as it is already discovered. Since both are both extremely difficult to fix and extremely difficult to implement, I'm not listing them here.
- **6 points:** If you find an exploit and give us a fix for it, you can receive 6 points extra credit upon verification of both the exploit and the fix.
- **1 or 2 points:** General simulator bugs are much more likely than exploits. Finding a bug and reporting it can get you 1 point. Fixing the bug and giving us the fix (you can't check it in directly but you can give it to us in the bug report) can get you two points. Both bug and fix must be verified for any extra credit to be awarded.

Competition ladder

The class-wide competition ladder will run every night from the day the project is handed out. Extra credit opportunities start 2 weeks before the project is due. For this project, each game will either be played against other teams (in a round robin fashion) or against hard-coded heuristic teams. Teams will be ranked by score given above.

The first place team will receive one extra point for each night that that team wins the ladder up to a maximum of 5 points. To win, you must be ranked above the Flag Collector/Random team. The second place player will receive 1/2 extra point for each night

that the player is in second place. No player can receive more than 5 extra credit points from the ladder and points will be distributed down the ladder accordingly should the maximum be reached.

Wanted (alive please): creative individuals

Creativity is highly encouraged! To make this real, there are up to 10 points of extra-credit available for creative solutions. Some ideas here include interesting and novel ways to incorporate previous approaches from the semester (including learning), partial order planning, or other approaches to multi-agent systems beyond planning and a shared knowledge representation.

- Document it in your writeup! I can't give extra credit unless I know you did something extra.
- Your agent must still be learning! Come see Dr McGovern for approval of any ideas. I'm not going to stop you from pursuing something you really want to try but I just want to make sure your project is doable.
- Remember that by being creative I am referring to the algorithm and *not* to the ability to creatively download code. All project code must be written exclusively by your group except for the sample players that we provide.

Option cards

I am sure the class can come up with some great new option card ideas and implementations. The list of all option cards provided in the board game is on D2L and we have only implemented a fraction of these so far. You can either choose to implement some of the remaining option cards or you can make up your own ideas! I know you are creative and will come up with some fun ideas! Any functioning option cards (you MUST include a unit test!) can be emailed to Dr McGovern for incorporation into the repository. Extra credit will be awarded for each such card.

Implementation details

All of your source code must reside in your src/4x4 directory and be in your 4x4 package. You may name your files within this package anything that makes sense to you (remember that we are grading on coding style as well). Using methods from the Java SDK is acceptable

(and encouraged as there are some nice built-in graph classes and a PriorityQueue class) but downloading or using code from any other sources is not allowed. See the syllabus for more details on what is considered academic misconduct. As discussed below, any additional files you create should be turned in along with your main agent class.

For this project, you will need to create a new TeamClient that extends the AbstractTeamClient class. There is an example team provided in RandomAndFCTeam. Instead of returning a single Program, this returns an array of Programs for each agent in the team.

Competing heuristics

For this project, you will play against the following heuristic team:

- **Flag Collector/Random team** This team consists of one Flag Collector agent and one random agent.

Those pesky details

1. Update your Roborally code from project 2. If you got subclipse working, you can update your code from within eclipse using Team → Update. If not, use the command line or tortoiseSvn to do "svn update". If you did not get the code checked out for project 0, follow the instructions to check out the code in that writeup.

2. Change the worldconfig.xml file in examples.robotest to point to your agent in src/4x4. The detailed instructions for this are in project 0. Make sure to copy over a clientinit.xml in the src/4x4 directory so your agent knows how to start. Because you will most likely want to run the ladder this time, also change the ladderconfig.xml and roboconfig.xml in examples.laddertest to point to your agent.

3. Create a team that uses planning and mulit-agent coordination as described above. Build and test your code using the ant compilation system within eclipse or using ant on the command line if you are not using eclipse (we highly recommend eclipse or another IDE!).

4. Test your player on a sample ladder. This will ensure that the agent runs on the CSN linux machines and that you have the agent correctly configured. To do this, we have created a submission script that will take your agent as input and run it against the heuristic agents several times and output the information to your terminal window.

WARNING: It will output a LOT of text so either be ready to scroll or cat the output to a file. To submit to this agent, do the following:

```
/opt/ai4013/bin/submit CS4013 Project3TestLadder *.java clientinit.xml
```

where you can simply list the java files instead of using *.java if you choose. Note that the xml file MUST be named clientinit.xml!

5. Submit your project on codd.cs.ou.edu using the submit script as described below.

   (a) Log into codd.cs.ou.edu using the account that was created for you for this class. Your username is your 4x4 and your default password is cs#4x4. Remember to change your password!

   (b) Make sure your working directory contains all the files you want to turn in. All files should live in the package 4x4. For example, if all of your code lives in `MyRoborallyAgent.java`, you would submit your code using the following command. The clientinit file is required to run your client!

   ```
   /opt/ai4013/bin/submit CS4013 Project3 MyRoborallyAgent.java clientinit.xml
   ```

   If you have extra code in `Foo.java` and `Bar.java`, you would submit using the following command:

   ```
   /opt/ai4013/bin/submit CS4013 Project2 *.java clientinit.xml
   ```

   (c) After the project deadline, the above command will not accept submissions. If you want to turn in your project late, use:

   ```
   /opt/ai4013/bin/submit CS4013 Project3Late *.java clientinit.xml
   ```

Point distribution

- 20 points for correctly implementing planning within your group's set of agents. A correct planner will choose among the high-level actions intelligently using ADL/STRIPS-style planning. Point distributions are given below:
  - 20 points for a fully correct planner.

  - 18 points if there is only one minor mistake. An example of a minor mistake would be not replanning at the correct time or incorrectly specifying the pre or post-conditions for an action.
  - 15 points if there are several minor mistakes.
  - 10 points if you have one major mistake. An example of a major mistake would be not using an pre or post conditions or incorrectly applying them to your state representation.
  - 5 points if you accidentally implement a search algorithm other than planning that at least moves the teams around the environment in an intelligent manner.

- 30 points for correctly implementing multi-agent coordination within your group's set of agents. A correct multi-agent systems will coordinate team behavior intelligently. Point distributions are given below:

  - 30 points for a fully functional multi-agent system.
  - 25 points if there is only one minor mistake. An example of a minor mistake would be not always coordinating well in a decentralized team or sometimes (rarely!) assigning incorrect roles in a centralized system. Note that errors in planning should count above rather than here. This focuses on the multi-agent coordination itself.
  - 20 points if there are several minor mistakes.
  - 15 points if you have one major mistake. An example of a major mistake would be not frequently failing to coordinate actions.
  - 10 points for several major mistakes.
  - 5 points if you somehow manage to coordinate occasionally but it is entirely accidental within the code

- 30 points for correctly implementing your chosen high-level behaviors. This will be graded as follows:

  - 30 points for making good use of your knowledge from AI so far to create good high-level behaviors
  - 25 points for one minor mistake. An example would be correctly designing a a "defend flag carrier" routine but implementing it such that you frequently bump into your carrier (causing loss of health).

  - 20 points for several minor mistakes or one major mistake. An example of a major mistake for the defend flag carrier behavior would be shooting your own flag carrier (on purpose as opposed to accidental bullets).
  - 15 points for two major mistakes
  - 10 points for implementing a behavior that is at least somewhat intelligent, even if it doesn't fulfill the requirements of the high-level behavior specified by your team.
  - 5 points for implementing a behavior that at least does something other than random movements.

- 10 points: We will randomly choose from one of the following good coding practices to grade for these 10 points. Note that this will be included on every project. Are your files well commented? Are your variable names descriptive (or are they all i, j, and k)? Do you make good use of classes and methods or is the entire project in one big flat file? This will be graded as follows:

  - 10 points for well commented code, descriptive variables names or making good use of classes and methods
  - 5 points if you have partially commented code, semi-descriptive variable names, or partial use of classes and methods
  - 0 points if you have no comments in your code, variables are obscurely named, or all your code is in a single flat method.

- 10 points for your writeup. The writeup must contain:

  - A full description of the approach (centralized or decentralized) chosen by your team
  - A full description of the high-level behaviors chosen by your team
  - A description of the goal condition
  - A specification of your state representation
  - A description of the pre and post-conditions on your actions.
  - Any additional information relevant to your choice of centralized versus decentralized (such as communication methods, etc).