Project 6
Computer Science 4013: Artificial Intelligence
Due 12:59PM May 3, 2011
Note: NOT the beginning of class

Introduction

The goal of this project is to create a team of agents that can cooperate to achieve a common goal. The agents initially are homogeneous but they can acquire capabilities that make them heterogeneous. The agents must cooperate to achieve the goal.

We will use the spacewar environment for this project. The SpaceWar problem domain takes its inspiration from the classic game of the same name written by Stephen Russell at MIT in the early 1960s. The original game is simple: a single star exerts a gravitational force on two ships in a toroidal space. A player can rotate her ship to the right or left, fire the thruster, lay a mine, raise shields, or shoot a cannon with the goal of hitting an opponent. The game continues until one of the ships is destroyed.

The version of SpaceWar that we're using in this course is modified from the original version. Instead of a central sun exerting a gravitational force, the environment contains some number of obstacles and gravity is neglected. Each ship carries an energy cell that fuels its thrusters, cannons, and life support systems. If the energy cell is depleted, the ship self-destructs. A ship can recharge its energy cell by collecting energy beacons from the environment, or by returning to its home base if it is available. Ships can also be damaged or destroyed by collisions with asteroids floating in space or from the cannon fire of other ships. Ships can compete individually or cooperate in teams.

For this task, you will play a modified version of the game Capture the Flag. The rules are summarized below.

- The goal of the game is to capture and take back to your base as many of your opponent's flags as possible during the 5 minutes of game play. To deposit a flag, you first need to unlock the base. Once the base is unlocked, you can deposit a flag by touching your own base with a ship holding the opponent's flag.

- Only one flag per team will be in play at any given moment. Flags are placed in one of your two alcoves, alternating locations after every time that a flag is captured. If you touch the opponent's flag, you pick it up. If you touch your own flag, it moves to the other alcove (no matter where it was). The flag will not be placed on top of the key.

- Only one key per team will be in play at any given moment. Keys will be available in the alcove opposite of the flags. Keys will behave like flags. If they are touched, they will swap alcoves. The key will not be placed on top of the flag.
- A base will only stay unlocked for a certain amount of time. Once it is locked again, you need to unlock it again with a key.
- A single ship cannot hold both a key and a flag.
- If a ship dies holding the flag or a key, the flag/key will appear where the ship died.
- Ships that die will respawn. The first respawn will be 3 seconds. It will take 2 seconds more each time that the ship dies, up to a maximum of 10 seconds.
- All ships can shoot and this includes your own team! Be careful who you shoot!
- All obstacles are stationary and non-destructible. The small ones are used to define your flag and base locations and the larger ones are a bumper of sorts around the world.
- A ship cannot shield while it has the flag or the key. If it selects the shield command in these situations, it will be ignored.
- Touching the base will only bring your energy up to 5000. Beacons give you a constant increase in energy. Your energy is unlimited, assuming you can touch sufficient beacons.
- The weapons available to the agent are: shooting energy missiles, repulser beams, tractor beams, electro-magnetic pulses, and mines. A ship can choose to shield itself temporarily or it can get out of the way of any incoming weapons fire!
- There will be a central command agent for each team. This agent cannot be shot at, moved, or killed! It exists outside of the spacewar arena and is used for multi-agent coordination.

Your task is outlined below. You will be using planning for part of the project as well as integrating many of the other techniques you have learned this semester. We will do some of the group work in class.

- **First**, create an agent that can move around the environment intelligently. You can use the provided A* code (see implementation details for more information on this) for this or you may write your own. This agent should avoid the asteroids and move

efficiently from one location to another. This will form the basis of many of your high level behaviors.

- Identify a set of high-level behaviors appropriate for each capture-the-flag team team member. Implement the set of high-level behaviors designed 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! Try to make effective use of the many types of weapons available to your ships as well as good use of the avoidance strategies.

- Implement planning within your team or within a single agent. I suggest that you implement it at the highest level (e.g. deciding which high-level behavior to choose at any time) but you could also use planning at a lower level such as improving A^* or other navigation functions or in deciding which weapon to use in a given situation. For each behavior used by your planner, specify a set of pre-conditions and post-conditions, appropriate for use in planning.

- Implement a central command agent that does effective multi-agent coordination. If you used planning at the highest level, this can be done using planning. Ensure that you re-plan as frequently as events warrant, such as a high-level action completing or sudden loss in energy. If you put planning at a lower level, then ensure that your agents are effectively coordinating their actions using another AI approach.

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: exploits in the simulator

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

- **5 points:** If you find an exploit and report it to us, you can receive 5 points extra credit upon verification of the exploit.

- **10 points:** If you find an exploit and give us a fix for it, you can receive 10 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 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 start on April 17. Teams will be ranked first by the average number of flags (rounded to the nearest tenth). Ties will be broken by an agent's average kills and then lifespan.

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. The second place team will receive 1/2 extra point for each night that the team is in second place. No team can receive more than 5 extra points and points will be distributed down the ladder accordingly should the maximum be reached.

The maximum clock time for any game will be 5 minutes. **Any team that takes more than 5 minutes for a game will be removed from the ladder for the remainder of that night.**

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: function approximation (this environment is perfect for function approximation) and hierarchical learning (e.g. learning to choose among the high-level behaviors and learning better low-level behaviors such as improved steering, shooting, or obstacle avoidance).

- Document it in your writeup! I can't very well give extra credit unless I know you did something extra.
- You must still be doing planning (speak to me if you want to verify that your approach is still planning).
- 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 you except for the sample players and code that we provide.

Implementation details

Be sure to edit your teamclientinit.xml found in examples/team-spacewar to point to your team client!

The ImmutableShip API has a getTeam() method which returns your team id. The getFlags() method tells you how many flags you have captured (as getBeacons() told you how many beacons you had captured). ImmutableSpacewarState gives you the state of the game. There are sample teams and sample agents in edu.ou.spacewar.mlfw.clients. Bases can be located through the state.getBases() call and flags with state.getFlags() and keys with state.getKeys(). As with previous projects, you will only have access to Immutable versions of each class (so you can't affect the game state).

To make navigation easier, we will put an existing spacewar A* agent on D2L. You can use this agent's code as a starting point for A* or you can write your own navigation function. At the very least, examine how the agent moves from one point to another as BeaconCollector only moves in a straight line fashion and the example agent moves more intelligently using pd-control.

Those pesky details

1. Update your code from the svn repository. Spacewar doesn't exist in your old repository! Your sample team client is in edu.ou.spacewar.mlfw.clients.RandomTeamClient. Sample agents are BeaconCollector (same directory) or the A* agent on D2L.

2. Create a ship that acts intelligently as described above.

3. Ensure that your player runs on linux on the CSN class machines. You can ssh into these machines or go into the lab personally to verify this. Java should be in your path by default. If it is not, java lives in:

   ```
   /opt/java/bin/java
   ```

4. ONE member of your team should 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.

   (b) Make sure your working directory contains all the files you want to turn in.

```
/opt/ai4013/bin/submit CS4013 Project6 teamclientinit.xml *.java
```

(c) To test your agent in the ladder, run the test ladder using the following command.

```
/opt/ai4013/bin/submit CS4013 Project6TestLadder teamclientinit.xml *.java
```

(d) 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 Project6Late teamclientinit.xml *.java
```

Point distribution

- 20 points for correctly implementing planning within your group's set of agents. A correct planner will choose among the actions intelligently using 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 high-level behaviors chosen by your team
  - A full description of the planning approach chosen by your team. This includes a full description of the goal condition, the state representation, and the preconditions and effects for each action.
  - A description of how you implemented multi-agent coordination