ICCS Coursework 2: Game AI

Zack Lyons

Brief

In this coursework you will be designing an Artificial Intelligence to play the game *Unreal Tournament 2004*. Using the set of behaviours provided, along with any further action or sense primitives you wish to implement, you will put together at least one POSH (Parallel-rooted, Ordered Slip-stack Hierarchical) plan which will be executed at run-time by your agents. Three bots will be running on your team; you should strongly consider giving each its own plan and working out how they will complement each other. Everything you need is installed on the lab computers in EB0.8; if you want to get this running on your personal machine you will need to acquire your own copy of *Unreal Tournament 2004*, which is available cheaply online. The tournament will be run on the lab computers.

Tournament

Your game AI will be assessed through a tournament (30%) and a written report (70%). The tournament will consist of two parts:

- **Thursday 3rd April (10:15 – 12:05)** – Group Stage. This lab is mandatory for everyone doing this coursework. You will be randomly placed in groups of 4 or 5 and asked to run matches against every other person in your group. Each team will have three bots whose goal is to capture the opponent's flag and protect their own. The winner of a match is the first to get three flags, or whoever is in the lead after ten minutes. After every group has finished their matches, the top person will progress to the next stage.
- **Tuesday 8th April (16:15-17:05)** – Direct Elimination. The winners of the group stage will compete against each other and be knocked out of the competition until one is left. This lab is not mandatory but it would be nice to see people coming along to watch these final matches.

Tournament Marking

You will get:

- 20% for competing in the tournament and adding in some new functionality (Hint: the tournament map will have objects that need to be jumped over).
- 5% for winning two or more games in the Group Stage.
- 5% for winning the tournament.

If the winner of the tournament has previous experience with POSH or Behaviour Trees (whether from last year or their dissertation project), the highest-scoring person without such an advantage will also receive a discretionary 5%.

Getting Started

1. Download the POSH UT2004 source code from the repository. Git and SmartGit are installed on all computers in EB0.8.
    a. Git command line: git clone https://code.google.com/p/posh-sharp/
    b. SmartGit GUI: Go to Project -> Clone and enter https://code.google.com/p/posh-sharp/ in the Remote Git Repository URL field. Click Next and select Git as repository type. Set the Local Directory path to somewhere on the C: drive (if you use the H: drive you will need to copy it to C: before you can run), then click Next. Enter a name for your project then click Finish.
2. Go to the source folder you have just cloned and look for POSH-sharp.sln. Right click and select Open With "Microsoft Visual Studio 2010" (not 2012). You may be asked if you want to open untrusted projects: click yes (they can be trusted!).
3. Right-click on the POSH-sharp project and click Properties. Go to the Debug tab and check that `-v -a=PoshBot.dll POSHBot` is in the Command Line Arguments field.
4. Right-click on the POSH-sharp.core project and click Build. Then do the same for POSHBot, and finally POSH-sharp.
5. Run *Unreal Tournament 2004* and select Host Game. Double-click on GameBots CTF Game as the game type, and then choose CTF-Bath-CW3-small for the map. Click Dedicated and a console window will appear. The server is now running and available to join. You can configure the server name (there will be lots of servers running on the LAN during this coursework) under [Engine.GameReplicationInfo] in C:\UT2004\System\UT2004.ini.
6. Run *Unreal Tournament 2004* again and this time select Join Game. Find your server from the LAN list and join it, then select Spectate.
7. In Visual Studio click Start Debugging. Your bot should appear in the game and can be observed from spectate mode. (The solution contains breakpoints for debugging but these can be toggled off.)
8. Finally go to http://code.google.com/p/abode-star/ and download "Abode-star vs 004" to a convenient location. ABODE is an IDE to be used for writing your bot's POSH plan.

The Coursework (Getting More Started)

After inevitably spending some time shooting the bot, you may realise that the bot could be smarter. This is where your coursework comes in. You will be improving bots in several ways: programmatically implementing actions and senses, and altering its plan.

1. Implementing actions and senses.
    a. In the POSHBot project you will see the Combat, Movement, Navigator and Status classes. These contain all the current actions and senses the bot has; spend some time having a look at these but **do not** modify them.
    b. Template.cs contains comments that explain the contents of these behaviour classes, and provides you with a template to make your own. Try to add a new action or sense.
    c. You may have as many additional classes as you like, each with any number of actions and senses. Changes to the pre-existing classes will not be used in the

tournament or marking. Your new behaviours will be automatically added to the bot's library the next time you compile POSHBot.

2. Modifying the plan.
    a. Open ABODE, then go to File -> Open and navigate to YourFolderName/POSHBot/plans. Select the .lap plan there and it will be displayed.
    b. After making changes, save as <username>.lap (e.g. ab123.lap).
    c. In Visual Studio, open POSHBot_init.txt in the POSH-sharp project (under library -> init). Change [PoshBot] to [<username>] and change the bot name to <username> A.
3. Making a bot team
    a. You can add more bots simply by adding more entries in the plan file. An example is commented out by default. You can specify the plan to be used by each bot by just changing [POSHBot] to the plan you want to use for that bot, e.g. [MyPlanA].
    b. Note that all the agents can share the same behaviour library (senses & actions), they will behave differently because of their individual memories (variable state) and individual motivations (plan file)

Notes

You are now ready to begin. When developing your bot, keep in mind the following (these will make more sense after you've gotten started):

- The CTF-Bath-CW3-small map is the most optimised for these bots, and is the most similar to the map that will be used in the tournament.
- You should check for the closest navpoint by using Navigator.close_navpoint(). This will give the current position of the bot.
- Do not hard code any logic that will make assumptions based on the node names. The node names will be changed for the tournament.
- You are not allowed to use GETPATH or the action Movement.SendGetPathHome(). You either have to copy and modify the path logic from Navigator or come up with your own method of pathfinding. The navpoints provide an interface for finding connected navpoints which is really useful and should be used. So using an engine call to get a PATH is not allowed, but getting the navpoint grid from GetBot().navpoints is. Look at navpoints from the bot's point of view.
- You cannot change team during a match or add items to your bot. Cheating will result in disqualification and your code **WILL** be checked! If you are unsure, just ask!
- The bot can send messages to the server using GetBot().SendIfNotPreviousMessage(<command>, <dictionary>);
- GameBots2004.pdf is a resource on Moodle that describes the protocol for communicating with the server and is really helpful for seeing what you can do.

Good luck!

This document is a help resource intended to explain everything you need to know to get your bot(s) working and competing. Usual Moodle and lab support will be available as always so if you are struggling just ask on the forums or in person.