ICCS Coursework 3: Game AI

Zack Lyons, Swen Gaudl & Joanna Bryson

Brief

In this coursework you will be designing Artificial Intelligence to allow game characters to play the game *Unreal Tournament 2004*. You will be provided with a set of behaviour modules that support a set of action and sensing primitives, and a prototype POSH (Parallel-rooted, Ordered Slip-stack Hierarchical) dynamic plan for doing run-time action selection between the behaviours. You will modify the plan, possibly incorporating additional primitives from additional behaviour module(s) which you write. Three bots will be running on your team; you should strongly consider giving each its own plan and working out how the plans (and bots) will complement each other. Everything you need is installed on the lab computers in EB0.10. If you want to get this coursework running on your personal machine you will need to acquire your own copy of *Unreal Tournament 2004*, which is available cheaply online. A final tournament will be run on the lab computers with the prize being a small number of points.

Tournament

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

- **Thursday 16rd April (16:15 – 18:05)** – Group Stage. This lab is mandatory for everyone. 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 players will progress to the next stage.
- **Tuesday 21st 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 support for these final matches would be fantastic, and a great turnout last year made for a brilliant atmosphere.

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 is a dissertation student with Dr. Bryson, or otherwise has previous experience of writing POSH plans, the highest ranking student without such experience will also receive 5%.

Requirements

To gain the 20% participation marks, you must:

- Add in some new functionality, including but not limited to the ability to jump over obstacles.
- Modify the existing plan or create a new one.
- Assign your plans to three bots.

A considerable advantage will be gained from:

- Finding and remembering the location of the rocket launcher. The rocket launcher will be on a fast respawn so all of your bots can get it.
- Giving your bots different plans.
- Having your bots communicate via team chat. How they communicate is up to you, but you might want to have them share paths, information about the flags, or locations of opponents or rocket launchers.

These features only serve to give you an advantage in the competition and in writing about your implementation. You should not spend more than the suggested hours on this coursework.

Setting Up

1. Download CTF-Bath-CW3-2015.ut2 from Moodle and copy it to C:\UT2004\Maps
2. Download the POSH UT2004 source code from the repository. Git and SmartGit are installed on all lab computers.
    a. Git command line: git clone https://github.com/suegy/posh-sharp.git
    b. SmartGit GUI: Go to Repository -> Clone and enter https://github.com/suegy/posh-sharp.git in the Remote Git Repository URL field. Click Next and Next again. Then set the Local Directory path to somewhere on the C: drive. (Important: running your code from H: drive will not work with UT2004. Desktop counts as H: drive). Then click Finish.
3. 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!).
4. You will see a list of projects in the solution explorer. Right click on the POSH.core project and click Build Solution. Then do the same for POSHBot, and finally POSH.
5. Go to C:\UT2004\System\ and open UT2004.exe. Select Host Game, select GameBots CTF Game as the game type, and choose CTF-Bath-CW3-2015 for the map. I suggest changing the game duration to something reasonable (long) otherwise you may find the map changing after 20 minutes. Click Listen to launch the server and spectate straight away, or Dedicated to launch a dedicated server (you'll need to run UT2004 again and select Join Game if you do this). You can change your server name under [Engine.GameReplicationInfo] in C:\UT2004\System\UT2004.ini
6. In Visual Studio go to Debug and Disable All Breakpoints. Then click Start Debugging. A bot named Celia (as demoed in lab) should appear in the game and can be observed from spectate mode.
7. Finally go to https://github.com/suegy/abode-star/wiki/Downloads and download Abode-star vs 004 to a convenient location. This is the IDE introduced in the lecture to write your plan.

Note that the C drive is a local machine drive, so you should save your relevant files to a USB stick or H drive after you finish using the computer. Do not rely on being able to use the same computer again! (Or that no one would change or copy your files.)

Starting Development

After shooting your bot or watching it walk into walls for a bit, you may realise it could be smarter. This is where you come in. You will be improving it in two ways: programmatically implementing actions and senses, and altering its plan.

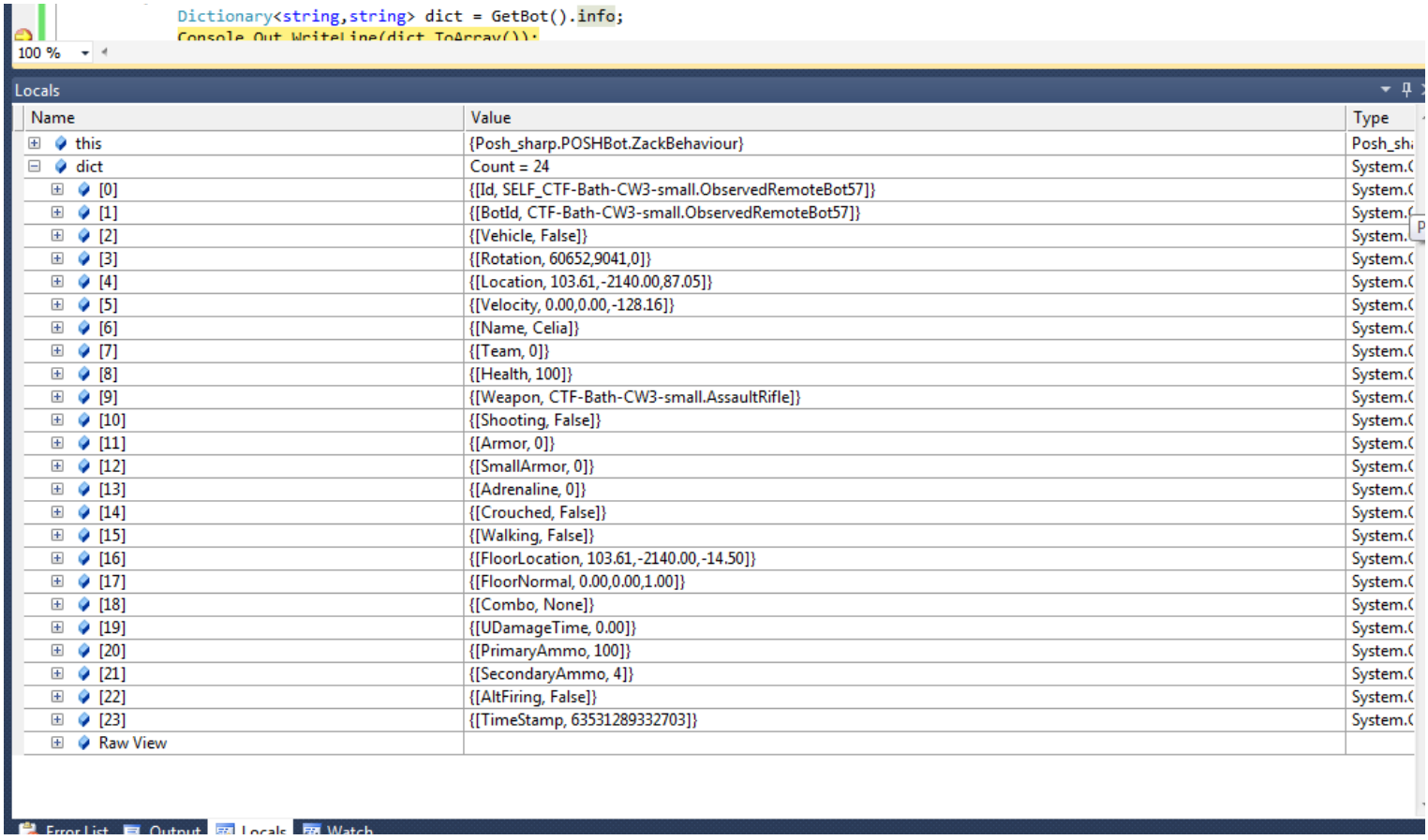
1. Implementing actions and senses.
    a. In the POSHBot project you will see the Combat, Movement, Navigator and Status classes. These are the behaviour modules, which contain all of the current action and sense primitives the bot has. Spend some time looking at these but ***do not*** modify them. Of course in BOD you would normally modify any module, but here we will be sharing libraries across many developers, so it's best to keep these ones the same. If you do want to modify an existing behaviour, make a subclass and modify that.
    b. Template.cs contains comments that explain the contents of these behaviour classes and provides you with a template to make your own. Duplicate Template.cs, naming it Username_description.cs, where "description" is something appropriate to describe the module. E.g. Zl221_jumping.cs. You may need to add it to the solution explorer by right clicking POSHBot and selecting Add -> Existing Item. Try to add a new action or sense in your class.
    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. This is not a regular feature of POSH and Behaviour-Oriented Design, just a requirement for this coursework. Your new behaviours will be 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. When you have made some changes, save as username_description.lap (e.g. zl221_attacking.lap).
    c. Navigate to YourFolderName/POSHBot/init and open POSHBot_init.txt. Change [POSHBot] to [username_description] (e.g. [zl221_attacking]) and change the bot name to something identifiable.
    d. You can add more bots by adding more entries in this file. 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.

Notes

You are now ready to begin. When developing your bot, keep in mind the following:

- The CTF-Bath-CW3-2015 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 a node name. 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. Attempts at cheating will result in disqualification from tournament-based points, and your code WILL be checked! If you are unsure, just ask!
- The bot can send messages to the server using GetBot().SendMessage(<command>, <dictionary>); Generally you will be using GetBot() for everything. Visual Studio has a very useful auto-completion feature to show you what you have access to.
- 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. Since there is a lot of content, here are the bot attributes you can access to get you started. You may find it easier to search GetBot() in Visual Studio than use the pdf.



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.