CS 480 Task 1: Rational Bird Agent

Description

This task serves as an introduction to the underlying computational aspects of the topics we are covering throughout this course. This early in the quarter, we do not know how to do "AI programming" yet. Nevertheless, we can already start with the foundation of intelligent agents and systems by loosely modeling their commonsense physical behaviors and constraints with our existing programming knowledge. In particular, we are defining a simple bird, which happily flies throughout its magnificent aerial world with no purpose doing nothing useful at all.

The second goal is to refresh your memory on basic algebra and trigonometry, which are prerequisites for this course. The math in this course is not difficult, but it causes many students endless difficulties. You are responsible for your own implementations of each task; use this first one to develop a set of reusable data structures and methods.

The final goal is for you to familiarize yourself with the graphical viewer we will be using throughout the quarter. (The terms "viewer" and "visualizer" are interchangeable.) The installation and usage instructions are at the end of this document.

Requirements

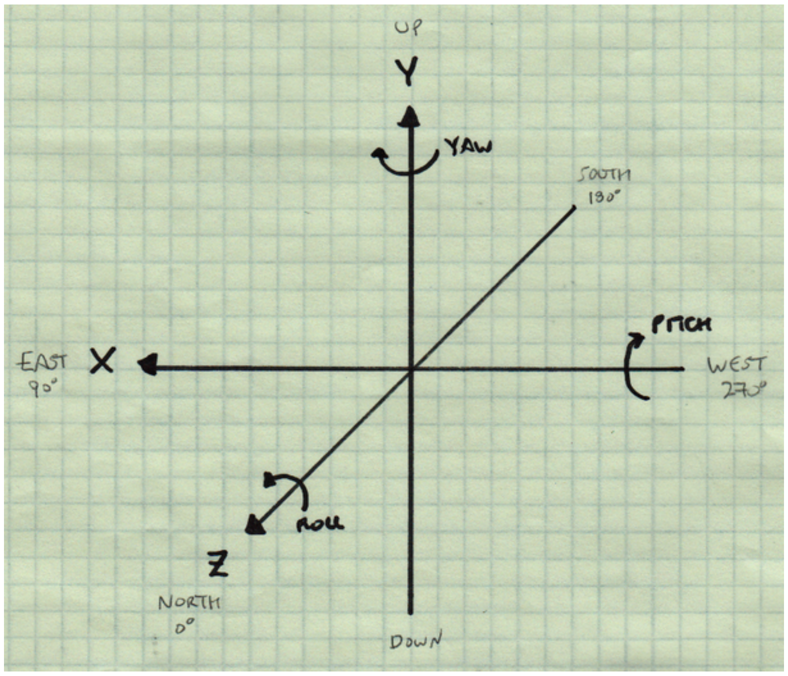
You must satisfy the following requirements in a single Java program (with one main method, but any number of classes) based on the provided `CS480Task1Template`. Be sure to comment your code so it is clear where each part is handled.

Section 1: Modeling

Part 1

Generate a series of 10 connected line segments distributed randomly within a three-dimensional aerial region. The start and end points of the series do not need to join.

The world is 1000 meters square, with the origin at (0,0,0). The coordinate system is defined as follows. It is a standard right-hand system used in graphics and many other places: if you point your index finger toward increasing X, your middle finger will point to increasing Z, and likewise your thumb to Y. Be careful because the equations you learned in math class may not directly translate to this system.



Part 2

Write an algorithm to follow the lines (the *course*) defined in Part 1. For example, if a line segment goes from (0,0,0) to (10,15,20), then your algorithm should generate the intermediate coordinates along this line, at each of which the bird will be over time. The interval between these coordinates is your choice, which you can determine empirically by looking at the visualization. Your bird knows the end points of each line segment, as well as their sequence in the series, so you are not doing any kind of image processing. In other words, the viewer plays no role in your calculations. Also note that the bird is not to scale. The bird does not need to face where it is going.

Part 3

Extend Part 2 to adjust the attitude (the *heading*) of the bird accordingly as it moves. For example, if the line goes north and somewhat upward, then the bird should be oriented to yaw 0 and perhaps pitch 30. The attitude coordinate system is defined by (*yaw*,*pitch*,*roll*), which respectively map onto the (y,x,z) axes in the position coordinate system. The ranges in degrees are [0,360), [–90,+90], and [–180,+180], where (0,0,0) is yawed north, pitched level, and rolled level, respectively. Increasing values result in clockwise (north to east to south to west), upward (pitched down to pitched up), and rightward (rolled left to rolled right) changes, respectively. Roll is irrelevant in this task, however, so it should always remain 0.

The order of operations does not matter in this task (i.e., pitch then yaw versus yaw then pitch), but it may in later tasks. Always make sure you understand the model you are controlling. We take the physical world for granted as simple, but translating it to the mathematical and programming world can be vexing.

Part 4

The course in Part 3 is too good to be considered realistic. In other words, no bird follows a perfect path (the *track*) over the course. Extend Part 3 to modulate the course with some kind of semi-realistic variation (*noise*). For example, you might consider imposing a weak sinusoidal wave over it. Your goal is to perturb the course, not to model any particular locomotion of the bird like flapping wings.

Part 5

Many of the random line segments will join at extreme angles. If your bird were to follow them literally (even with Part 4 in place), it would make highly unrealistic, instantaneous changes in position and attitude. Smooth the transitions to generate a more realistic track. For most angles, your bird will probably never actually visit the vertex. In some cases, it may have to deviate substantially to negotiate a turn. As long as your solution satisfies our “TLAR” test, consider it valid.

Section 2: Summary of Approach

There are relatively few well-defined solutions to AI problems, which means that your implementation can be anything you want. I will evaluate it primarily on its performance and your explanation of how it works.

For each part in Section 1, describe at minimum the following:

- what you did
- how well it worked
- what its pros and cons are
- what you might improve

This summary does not need to be extensive. A few sentences for each bullet should be adequate. Be honest: your summary must correspond to the actual performance of your model.

Submission

Submit your source file(s), a text readme file that explains how to compile and run your solution, and the summary document in PDF format through the link on the course web page.

For each part in Section 1, submit a representative track file of your output called `track1_`*`n`*`.trk`, where 1 is the task number and *n* is the section number. The track file is generated automatically by the viewer from the state data you send it. You must supply the filename in the constructor call to `Viewer`.

Grading

10 Section 1, Part 1
10 Section 1, Part 2
10 Section 1, Part 3
10 Section 1, Part 4
10 Section 1, Part 5
20 Section 2

<u>JOGL Installation</u>

There should be nothing to install for your operating system. The JAR and library files are part of the viewer setup:

Create a library folder somewhere on your machine and unzip the appropriate files (jars and libraries) from `shelby.ewu.edu/viewer/` based on your system. There are zip files for Windows 32-bit and 64-bit, Linux 32-bit and 64-bit, and OS X. Be aware that the appropriate files may not correspond to the size of your hardware architecture. For example, my office machine is 64 bits, but the 32-bit files were required. If one does not work, try the other.

In case you need a reference, JogAmp is documented at jogamp.org. Be forewarned that their documentation sucks.

Your Java source should not matter anymore: Oracle or open source should be equivalent for our purposes. But you probably still need to compile against Java 8 (aka 1.8) because the viewer uses this version (as do all classes in the CS program). If you get an error to the effect of "major.minor version conflict", it is for this reason.

<u>Task Configuration</u>

Every task will follow the same framework, but it will use a different template class and viewer JAR. (The JOGL files remain the same.) Always make sure you are using the correct version for the task. Occasionally, an updated version will be posted after the task is assigned, which I will announce.

Eclipse

The exact process may differ depending on your Eclipse version and OS. In general, the same steps should apply for any IDE. Connecting to the two sets of JAR files is the only trick.

These instructions may change slightly between tasks, but in general, the only difference should be in the task number.

Create a new project:

Select File→New→Java Project→Project Name→`CS480Task1`

Click Next

Click Libraries→Add External JARs→`cs480viewer-task1.jar` to access the current viewer from your code.

Also add all the JAR files from your library folder above so the viewer can access graphics on your system.

Add the task template to the project:

Copy `CS480Task1Template.java` to your workspace folder with your file browser.

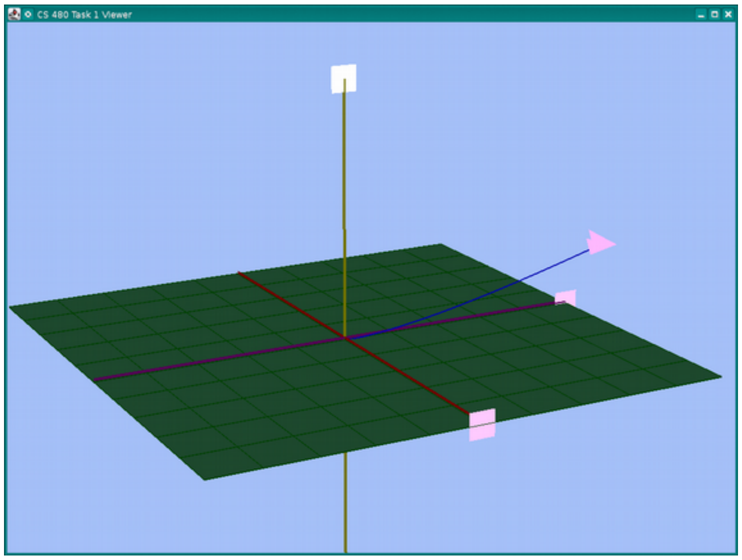
On Linux mine is `/home/dtappan/workspace/CS480Task1/src/`

On Windows it is `c:\users\dtappan\workspace\CS480Task1\src\`

Select the project in the Package Explorer and press F5 to refresh it. The template source file should appear in the list.

Compile and run the template (with the green triangle) to make sure everything is set up correctly. You should get no errors.

You should see something like this, with a bird-like triangle flying to the edge of the world. The next update (still for this task) will improve the cosmetics of the display, but this primitive form is still complete useable.


CS 480 Task 1 Viewer