Assignment #2: Multi-Player Games with 3P Cameras

Due: Sunday October 18th (3 weeks)

The objective of this assignment is to learn about third-person cameras, display systems, and two-player "split-screen" games, by upgrading your *"Dolphin Explorer"* game from A1, to become a new game called *"Dolphin Contest"*. Your game will now provide each player with their own *viewport* (on a shared screen), each will have their own dolphin *avatar*, and separate controls for third-person cameras. The "game world" consists of the two dolphins and other objects as before, but also a flat ground plane (e.g., the Y=0 plane) on which the objects reside nearby, and on which each player manipulates his/her avatar. The game also provides two HUD elements, one in each viewport.

The planets now must all reside on or near the ground plane. Initially they are unmoving. When a planet is visited by a player, it starts moving in place (such as rotating, or bobbing up and down). Players gain points by being the first to visit a planet. You may add other rules as you wish.

Note there are now TWO dolphins, one for each player. You may add other features, or even change the game, as long as your program satisfies the implementation details described below. Make sure that any changes you make to the game are described in the "Player's Guide" that you also submit.

Game Program Requirements

- Implement a 3rd-person Camera Controller, either a chase camera or an orbit camera. It may be simplest to implement an orbit camera, since we went over that in class in fair detail. If you implement an orbit camera, it must be possible to: (1) orbit the camera without altering the avatar's heading, (2) move and turn the dolphin while maintaining the camera's relative position and orientation relative to the dolphin, (3) adjust the camera elevation angle, and (4) zoom the camera in and out on the avatar's location. If you prefer to implement a chase camera, it must include some sort of "spring system" so that it moves smoothly behind the avatar when the player changes direction (implementing a spring system would require some additional research).
- Support split-screen multi-player viewports. Each of the (two) players must be presented with their own view of the (same) world which changes as their cameras move.
- Provide HUDs for each player, each displaying data relevant only to that player. Both HUDs must maintain their relative position within their respective windows if the player resizes the window.
- Include a *ground plane* – a flat surface on which the game takes place. You may if you wish allow players to leave the ground for a time ("jumping"), but they should not have the ability to move continuously in arbitrary 3D directions like they did in A1, and should not be able to go under the ground surface. You will need to build the plane out of two large triangles (each triangle is a `ManualObject`). You may also want to apply **`scale()`** and **`rotate()`** to adjust its size and position. Implementing the XYZ axes is not required.
- Two different types of *Node controllers*, at least one of which is written from scratch by you. One of the controllers is applied to a planet when it is first visited by one player, while the second controller is applied to a planet if it is first visited by the other player. Each node controller can do whatever you like - for example, it could cause the planet to *spin* or *bounce*. The specifics are up to you, but at least one of the controllers must be implemented by you, by creating a class that extends the RAGE **`AbstractController`** class.

- Support two input controllers. For example, one player might use the keyboard and mouse while the second uses a gamepad. *Document all of your player controls in the Player's Guide!*
- Your game must include at least one example of a hierarchical relationship between two or more scenenode objects in your game. That is, there must be at least one node (other than the dolphin or camera) that has a "child" node. For example, you might organize all of the planets as children of a node that you create, and then apply some node controller to the parent node (and thus to all of the planets) after all of them have been visited. Or you could make a hierarchical object.
- In this version of the game, there is nothing stopping the player from moving the camera (or the dolphin) completely off the ground plane – although your code should prevent them from going below the ground plane. There is also nothing stopping the player from positioning the dolphin avatar or the camera in such a way that some other object is blocking the camera's view of the avatar. That is ok for now – we will fix this (at least partially) in Project 3.
- Note that it is no longer a requirement that you can "ride" the dolphin. In this game, the dolphin is a player's avatar, and they watch it from a 3P camera. You should remove your existing code that hops off the dolphin to visit planets – your game will be more attractive if you remove this code and simply have the dolphin visit planets.
- The camera controller that you build can be constructed using either "c" mode or "n" mode. You will probably find it easier to use "n" mode since that is what is described in the code handout.

Deliverables

This is an individual assignment. Submit to Canvas exactly TWO items: (1) a TEXT (.txt) file indicating on which lab machine you tested your program, and (2) a *single ZIP file* containing:

1. your Java source code (`.java`) files
2. your compiled (`.class`) files for each of your Java source files
3. your assets folder
4. a batch (`.bat`) file that successfully runs your submission if unzipped as-is on a lab machine
5. a short "Player's Guide" document (approximately 2 pages, preferably PDF), including:
   - your name
   - a screenshot (JPG file) showing a typical scene from your game
   - how to compile and run your program from a command window
   - how your game is played, a list of the inputs and what they do, and the scoring
   - a description of your node controllers and what they do
   - a description of your use of group/child node relationship(s)
   - a description of your camera control, and whether it is an orbit or chase controller
   - a clear list of which requirements you *weren't* able to get working
   - a list of anything special that you added beyond what was specified in the requirements
   - which 5029 lab machine(s) your program was tested on. Be specific.
   - a list of every asset used in your game, and whether you made it. For each asset that you didn't create yourself, indicate where you got it, and provide clear evidence that it is legal for you to use in this game (such as written permission, or a posted license).

Note that the submitted files must be organized in the proper hierarchy *in the ZIP file*, as indicated in the coding style requirements given in assignment #1, but now in a folder called "**`a2`**".