Assignment #1: Building a Game using Game Engine Components

Due: Sunday night Sept 27th, at midnight

In this assignment you will use and extend game engine components to build a simple 3D game. The game engine will supply some of the underlying functionality, and you will implement some of the functionality yourself. You will use the CSc-165 fork of *Ray's Awesome Game Engine* (RAGE165).

The game is called *"Dolphin Explorer"*. The player uses the keyboard and gamepad (Xbox controller or equivalent) to fly around in outer space on the back of a dolphin, visiting planets. The planets are scattered randomly around the space, and to visit one, a player has to ride the dolphin up close to the planet, get off of the dolphin, and run into it (i.e., when the camera gets close to the planet – there is no avatar). As planets are visited, the player's score is incremented. The player is never allowed to stray too far from the dolphin. The game score is shown on a HUD (Heads-Up Display).

Game Program Requirements

- Your game must extend the RAGE class **`ray.rage.game.VariableFrameRateGame`**, and override **`setupCameras()`**, **`setupScene()`**, and **`update()`**.
- You should instantiate some planets (at least three), randomly positioned in 3D space. Use **`scale()`** to set the planets to various sizes. You can use the "sphere.obj" model to build the planets, and texture them using textures provided in the assets folder, or make your own textures.
- You must write classes that implement **ray.input.action.Action**, and then utilize **`InputManager.associateAction()`** to link device components with your Action classes.
- At least the following key mappings must be provided (you can add more if you like):
  - **A / D / W / S**: move camera *left / right / forward / backward*. When the player is ON the dolphin, these keys should move the dolphin. When the player is OFF the dolphin, these keys should move the camera by using a RAGE camera's U/V/N axes.
  - **Left-arrow / Right-arrow**: rotate camera (or dolphin) around the world up axis ("global yaw").
  - **Up-arrow / Down-arrow**: rotate camera (or dolphin) around its U side axis ("local pitch").
  - **SpaceBar**: toggle between being ON the dolphin, and OFF the dolphin. When using this key to hop OFF the dolphin, it should move the camera axes and position near the dolphin.
- At least the following controller mappings must be provided (you can add more if you like):
  - **X-axis**: move camera/dolphin left and right in the same manner as the A and S keys
  - **Y-axis**: move camera/dolphin forward and backward in the same manner as the W and D keys
  - **RX-axis**: yaw camera/dolphin around its V axis, the same as the Left/Right arrow keys
  - **RY-axis**: pitch camera/dolphin around the world up axis, the same as the Up/Down arrow keys
- The game must display X, Y, and Z **world axes** showing where the world origin is located. You can build these axes as ManualObjects out of two vertices, rendering as LINE.
- The game must include a **HUD** that shows at least the "score" of how many planets the player visited. You may also include other information in the HUD if you wish.
- Something in your game must keep the player (the camera) from moving too far from the dolphin.

- The player must be OFF of the dolphin in order to visit a planet and get credit for it.
- Your game should include ambient light, and at least one positional light.
- Two additional features of your own choice:
  - an additional game activity not listed above. For example, the player could collect a souvenir from each visited planet. The added activity is *your choice*, so long as it isn't too trivial.
  - an additional game object, built using RAGE's **ManualObject** class. You'll need to hand-build it and add it into your game, including the vertices, texture coordinates, indices, and normal vectors. You should try and build a shape that fits with your game's theme, although it isn't expected to be very realistic. For example, you could have a storage bin where souvenirs from each planet are collected. Or, you could add a monster that attacks you when you are off of the dolphin. These are just a few ideas, try to make up your own idea!

Coding Style Requirements

- Your code should be organized in two Java packages, one called "**a1**" to hold your game, and one called "**myGameEngine**" to hold your extensions to the RAGE game engine. In this assignment, it would be reasonable to put your code that moves the camera around (the Action classes) into the myGameEngine folder, since a camera controller is usually provided by a game engine.
- Your program must run correctly when invoked from a command prompt using the command **`java a1.MyGame`**. This means your **`MyGame`** class must be contained in the "**`a1`**" package and that it must contain the required **`public static void main(String [] args)`**. You can use the **`main()`** and constructor organization shown in the examples, if you wish.
- Your program must not crash if a gamepad isn't plugged in. If there is no gamepad, it should continue be playable using the keyboard.

Additional Notes

- In RAGE, cameras are always attached to a node, and can also be put into one of two modes:
  - ✓ **"`c`"** mode means the camera is controlled directly using its U, V, and N axes.
  - ✓ **"`n`"** mode means it is controlled by moving the node that the camera is attached to.

  When the player is OFF of the dolphin, set the camera mode to "`c`", and control the camera actions yourself via its U, V, and N axes. When the player is ON the dolphin, set the camera mode to "`n`", make sure the camera Entity is attached to the dolphin's Node, and move the dolphin around. You'll need to keep track of whether the camera is ON or OFF of the dolphin.
- When getting OFF the dolphin, in addition to putting the camera in "`c`" mode, you'll need to move the camera position and its U,V,N axes to someplace near the dolphin.
- When getting ON the dolphin, put the camera in "`n`" mode, and make sure the camera is attached to the dolphin's Node. You can make the player's view from the dolphin look better by adding a child Node to the dolphin, positioned slightly above the dolphin, and attach the camera to that.
- Planets are "visited" by running into them with the camera (when OFF the dolphin). This means you must do some simple collision detection between the camera location and the planets. For this assignment, you can simply iterate through the planet objects and check their distance to the camera. Later in the semester we will study collision detection in a more accurate manner.

- Your camera and dolphin movements must be computed based on elapsed time, so that their speed is the same regardless of which machine is being used. For this you can use the elapsed time available from the engine in RAGE (passed into update), as shown in the sample code.
- If you want to have other objects in your world move around, you can try using the built-in RAGE controllers for doing this. Or you can try retrieving an object's position and rotation vectors, and changing them. If you try the latter, you should notice that each node has "local" and "world" transforms. Your program is free to alter the "local" transforms, but don't mess with the "world" transforms (we'll learn more about this later).
- VERY IMPORTANT ➔ Make sure that your program works on at least one workstation in the RVR-5029 lab. Don't wait until the last minute to test this!!! The same goes for being able to run your program from the command line. I have no easy way of grading your homework if I can't easily launch it from at least one of the lab machines!

Deliverables

This is an individual assignment.

Submit to Canvas **TWO files (.zip file and .txt file) SEPARATELY** (i.e., do NOT place the .txt file inside the .zip file).

- The ZIP file should be named as YourLastName-YourFirstName-a#.zip (e.g., Doe-Jane-a1.zip) and should contain:

  (1) Java source code (`.java`) files,
  (2) compiled (`.class`) files,
  (3) a batch (`.bat`) file that runs your submission if unzipped as-is onto a lab machine, and
  (4) a short "Player's Guide" document (approximately 2 pages long **pdf** document), 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 additional "game activity"
  - ➢ a description of your additional "game object"
  - ➢ 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
  - ➢ 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).

  The submitted files must be organized in the proper hierarchy in the ZIP file as indicated in the coding style requirements listed above.

- The TEXT file **(i.e., not a pdf, doc etc.)** should be named as *readme.txt* and should list: **the name of the RVR 5029 lab machine you have used to test your program** (as indicated above, it is a requirement that your program runs properly on at least one machine in the lab). You may also include additional information you want to share with the grader in this text file. You will receive the grader comments on your text file when grades are posted.