Assignment #1: Building a Game and Extending a Game Engine

"Dolphin Adventure 1" ----- Due: Friday Feb 17th (3 weeks)

The objective of this assignment is to learn how to use and extend game engine components to build a very simple 3D game. You will be relying on existing game engine components to supply much of the underlying functionality, while implementing some of the functionality yourself. The assignment requires you to use – and modify – your own copy of the TAGE game engine distributed in class.

The game you will make is called *"Dolphin Adventure 1"*. The player uses the keyboard and gamepad (Xbox controller or equivalent) to fly around in outer space on the back of a dolphin, collecting prizes. The prizes are scattered randomly around the space, and to visit one, a player has to ride the dolphin up close to the prize, get off of the dolphin, and run into it (i.e., when the camera gets very close to the prize). This causes the prize to be collected, and the score incremented. The player (camera) is never allowed to stray too far from the dolphin. The score is shown on a HUD (Heads-Up Display).

Game Program Requirements

- Your game extends **`VariableFrameRateGame`**, and overrides at least **`loadShapes()`**, **`loadTextures()`**, **`buildObjects()`**, **`initializeLights()`**, **`initializeGame()`**, and **`update()`**. The game's logic is to reside in **`update()`**, or in functions invoked by **`update()`**.
- Your scene should instantiate some number of "prizes" (at least three), randomly positioned in the 3D space. You should also use **`scale()`** to set the prizes to various sizes. You should use the built-in shapes for the prizes, and texture them. At least one of the textures must be made by you.
- Your game must support input actions handled by **Action** classes. That is, you must write classes that extend **AbstractInputAction**, 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**: turn *left / right* (yaw). When the player is ON the dolphin, these keys turn the dolphin. When the player is OFF the dolphin, they turn the camera by using the camera's UVN axes.
  - **W / S**: move *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.
  - **Up-arrow / Down-arrow**: rotate the camera (or dolphin) around its U side axis ("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**: yaw (turn) the camera/dolphin left and right in the same manner as the A and D keys
  - **Y-axis**: move camera/dolphin forward and backward in the same manner as the W and S keys
  - **RY-axis**: pitch camera/dolphin around its U axis, the same as the Up/Down arrow keys
- New *yaw()* and *pitch()* functions should be created and added to the **GameObject** and **Camera** classes, and used for the inputs described above. *Global yaw* is recommended but not required.
- The game must display X, Y, and Z **world axes** showing where the world origin is located. You can build these axes as Line shapes.

- The game must include a **HUD** that shows at least the "score" of how many prizes the player collects. You may also include other information in the HUD if you wish.
- Your program must keep the player (the camera) from moving too far from the dolphin, such as b not allowing a move if the resulting distance would be more than some threshold.
- The player must be OFF of the dolphin in order to visit a prize 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 dolphin could be required to periodically eat some food. The added activity is *your choice*, so long as it isn't too trivial.
  - an additional game object, built using TAGE's **ManualObject** class. You'll need to hand-build it, including the vertices, texture coordinates, and normal vectors. 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 the prizes are collected. Or, you could add a monster that attacks you when you are off of the dolphin. These are just ideas, try to make up your own!

Additional Notes

- You'll need to keep track of whether the camera is ON or OFF of the dolphin. When it is ON the dolphin, your program will need to update the camera location/orientation as the dolphin moves.
- When getting OFF the dolphin, your program should move the camera position to an appropriate location near the dolphin, and facing the same direction as the dolphin.
- Prizes are "collected" 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 prizes. For this assignment, you can simply check each prize object to see if its distance to the camera is sufficiently small. Later in the semester we will do more sophisticated collision detection.
- If you don't have a controller that implements X, Y, RX, and RY axes, check with your instructor – he has just a few extras available for checkout. Cheap ones can be purchased for about $25 so you should consider getting one for yourself if you don't already have one.
- 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. The posted examples do not always do this, so you should experiment with using the time elapsed since the previous frame.
- If you want to have other objects in your world move around, you can use the built-in TAGE rotation controller, 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. You can use the "world" transforms to look up the position and orientation of an object. You can change an object's position or orientation by altering the "local" transforms, but do not modify 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 and Coding Style Requirements

- This is an INDIVIDUAL assignment.
- Submit to Canvas TWO items:
  - (1) a ZIP folder with your A1 game submission and game engine, as described below
  - (2) a TEXT file (.txt) indicating which RVR-5029 lab machine you tested your program on
- The ZIP folder must be named with your full name, as *lastname_firstname*. Inside this folder are:
  - ✓ (1) a subfolder "**tage**" that is the package containing your version of TAGE
  - ✓ (2) a subfolder "**a1**" that is the package containing your game application
  - ✓ (3) the "**assets**" subfolder
  - ✓ (4) batch files "**compile.bat**" and "**run.bat**"
  - ✓ (5) a PDF **readme** document.
- Your game code is to be placed in the "**a1**" folder, which is the Java package for your game (do not call this package folder "myGame" as was done in the sample program). Your main program that extends VariableFrameRateGame should be in a file inside of a1 named **myGame.java**.
- The "**tage**" folder contains the complete TAGE game engine, including your changes. Your new *pitch()* and *yaw()* functions should, for example, be found in the GameObject and Camera classes. (It is NOT necessary for the "tage" folder to include any of the `.jar` files that TAGE relies on.)
- You are encouraged to make other changes and additions to your copy of TAGE – however you should document any such changes in your **readme** PDF file.
- Your `compile.bat` file should successfully compile BOTH your TAGE engine, and your a1 game application.
- Your `run.bat` file should successfully run your game, using the following command:

```
java -Dsun.java2d.d3d=false -Dsun.java2d.uiScale=1 a1.MyGame
```

- Your "**readme.pdf**" document (approx. 2 pages) must include the following 9 numbered items:
  1. your full name and CSc-165 section number, and "A1 – Dolphin Adventure 1"
  2. a screenshot (JPG file) showing a typical scene from your game
  3. how your game is played, a list of the inputs and what they do, and the scoring
  4. a description of your additional "game activity"
  5. a description of your additional "game object"
  6. a clear list of all changes you made to the TAGE engine
  7. a list of any requirements that you *weren't* able to get working
  8. a list of anything special that you added beyond what was specified in the requirements
  9. 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). If an asset was copied from the distributed TAGE examples, just state that.

Note that the submitted files must be organized in the proper hierarchy *in the ZIP file*, as indicated in the coding style requirements listed above, *except* the TEXT file which must be *outside* the ZIP folder.