Assignment #3: Game Project

Due Date: See *Deliverables*, below

The remainder of the semester will be spent developing your own video game. Guidelines and requirements for the game implementation are given below. The end result will be a completed game which you will demonstrate (and everyone else will get to play!) at the end of semester "Game Party".

The project is to be done in groups of two. If you insist on working alone, this will be allowed. But the assignment is designed assuming it would benefit from two students working together on it. The requirements and quality expectations are the same if you work alone or with a partner.

Project Requirements

Your game must include each of the following features. A few of these are topics we have already covered, but most of them are things we will be going over in the coming weeks.

- **External models**. The game must include at least two custom-made models designed by you, *Each person in the group must contribute at least one of those external models.* Blender or Maya are recommended. Note that RAGE supports only "OBJ".

  (related) – Each of these two models must utilize a skin (texture) of your own design using UV-unwrapping, and the texture must be properly applied in your game.

- **Networked Multi-player**. Your game must be able to be played by multiple players (at least two) over a network. It is a requirement that players be able to see each other's avatars in their view. Note: it is *also* a requirement that your game function in *single-player mode* (see below).

  (related) - Players must have a way of selecting their character avatar model at startup, and the selection must be conveyed to other game clients so their ghosts use the proper models for each player. This means if you use the simple game network protocol discussed in class, it must be extended to pass a model name as part of the CREATE message sent to remote clients.

- **Scripting.** The way in which you incorporate scripting is up to you; it could be used for configuration/initialization, for dynamic world and/or algorithm modification, or something else. JavaScript is the easiest since it is built in to Java, but you may use Python or Lua if you prefer.

- **Skybox and Terrain**. Your game must include a *skybox* and *terrain*. You may use the RAGE Skybox and Terrain classes, or you may develop your own. Note that it is not a requirement that these components be integral to the gameplay. For example, if your game takes place mostly "indoors" then you can meet this requirement by providing a way for a player to "go outside" and move around on (and up and down on) the terrain, bounded by a skybox, which has no direct effect on the game – but the program must at least demonstrate use of a skybox and a terrain.

- **Lights**. You must utilize the RAGE lighting classes somewhere in your game. In particular, you must make *effective* use of at least two lights (positional, spotlight, or directional) in addition to the general ambient light. At least one must be possible for the player to turn on and off.

- **3D Sound**. Your game must support several 3D action-specific game sounds as well as background sound. It is recommended that you use RAGE's audio package (which uses JOAL), or you may use JOAL directly, or you may use another 3D sound package if you want.

- **HUD**. Include a HUD component to let the player know what is happening in the game. If your game is better without a HUD, you may include an input to enable/disable the HUD display.
- **Hierarchical SceneGraph**. There must be some portion of your scenegraph that utilizes hierarchical transforms. It could be a hierarchical object, or a hierarchical system of objects (such as a planetary system). You should use RAGE's scenegraph support for this.
- **Animation**. Add skeletal animation using keyframes for at least one model in your game. It does not have to be complex, but should look reasonably smooth. The animation must be created by you, using Blender. You can then use RAGE's Blender export tool and import it into your game.
- **NPCs**. Your game must include one or more "Non-Player Characters" managed by at least a rudimentary AI controller. This means the NPCs must do something under AI control, not just stand around frozen, and not just move randomly. We will study various approaches in class.
- **Physics**. Your game must include some use of a *physics engine.* You may use RAGE's built-in physics package (it uses JBullet), or you can use a physics package (such as JBullet) directly. You can satisfy this requirement anywhere in your game, for any reasonable purpose.

Additional Notes

- Your game may be based on first-person (1P) or third-person (3P) view. In either case, player avatars must be visible in remote client views.
- Your game must support both FSEM (full screen exclusive mode) and windowed mode.
- There must not be any hard-coded IP addresses in your code. You may instead require the server's IP to be provided on the command line, or you may prompt for it interactively. If you wish to use a different form of server discovery, ask your instructor.
- You may use any network architecture you like for your game. A client-server "fat-client" approach with UDP or TCP, using the RAGE networking package, is likely to be the easiest.
- You are encouraged to add additional models (beyond the two you are required to create yourself), taken from any source. Make sure that they are in the public domain, or that you have permission to use them. In either case, your report must provide evidence that they can be used – such as a copy/paste of the license, or email message from the creator.
- It is a requirement that your game have a "single-player mode", even if it is designed as a fully multi-player game. Note that it is *not* a requirement that the game be very logical in this mode; for example, it's not necessary that there be a way to "win" in single-player mode – only that a single player be able to run around in the world, see the above features, and interact with NPCs. Your game would not include *ghost* avatars when in "single-player mode".
- All resources required by your game (e.g. texture files, sound files, models, etc.) must be accessed using *relative-paths* (that is, don't hard code absolute path names into your program). Also, be sure to submit all the required resources with your program (see "Deliverables", below).
- Your program must run correctly on at least TWO of the machines in RVR 5029 (it's networked!)
- You may change the name of the starter package/class to something matching your game's name (rather than the generic "**`a3.MyGame`**"). *Document the package name in your submission*.

When satisfying the above requirements, don't just copy the examples from class. In each case, try to find ways of using the techniques in a manner that fits *your* game and its theme.

Deliverables

There are several deliverables for the Project, including progress *Milestones*, each with its own due date and requirements. Each team arranges to meet with the instructor (or grader) to demonstrate completion of Milestones during the indicated weeks. Scores will be assigned based on satisfactory progress (meaning solidly working, but possibly not in its final form) of each Milestone separately.

Milestone 1: Skybox, Terrain, Scripting, & Networking. Week of April 2 - 5.

Milestone 2: External Models, Skinning, Physics. Week of April 15 - 19.

Milestone 3: Sound, NPCs/AI & Animation. Week of Apr 29 – May 3.

Game Party: Be prepared to give a demonstration of your working game at our GAME PARTY (scheduled during Finals week, exact date/time TBD). Each team will have 5 minutes to show the game in action. Then we take turns playing each other's games!

Final Code and Report. DUE: Saturday, May 11th at Midnight. BOTH team members are to submit to Canvas the same identical single ZIP file containing the following set of deliverables:

- Both the source code (.java) and compiled (.class) files for your game;
- A batch (.bat) file, or set of batch files, for running your game. You may provide separate batch files for starting the server, and for starting clients;
- A folder containing all the resources required by the game (texture maps, sound files, model files, etc.), *organized in the zip file in the way they need to be accessed by the game* (this can be satisfied by utilizing the *assets* folder as we did in A#1 and A#2);
- A guide to your game (PDF preferred) with the following 12 numbered sections, in this order:
  1. the name of your game, and your names
  2. at least one image (screenshot) showing a typical scene from your game being played
  3. how to compile and run your game
  4. any special device requirements, such as a particular video card or input device(s)
  5. how to play your game, including what things happen and how the scoring works
  6. what player controls are available (what keyboard keys do, etc.)
  7. how, and for what purpose, it uses scripting
  8. a statement indicating the (1) *genre*, (2) *theme*, (3) *dimensionality*, and (4) *activities* utilized in your game (see "Course Introduction" notes for examples)
  9. an explanation of where in your game each requirement listed above is satisfied
  10. any of the requirements that you *weren't* able to get working
  11. any technique you used in your game that goes beyond the requirements
  12. the contributions of each team member, including who designed which model(s)
  13. a list of items you created yourself (models, textures, heightmap, etc.)
  14. evidence of permission to use any item (models, textures, etc.) not listed in #13
  15. which RVR-5029 lab machine**s** (at least two – it's networked!) on which your program was tested and is known to work correctly on.

All submitted work, other than the use of public-domain models as described above, must be strictly your own or that of your team partner.

APPENDIX A

Previous requirements, and how they apply in this project:

- Constructing a manual object from scratch. Not required.
- Instantiating built-in RAGE objects/assets (such as the dolphin). Not required, but allowed.
- Input actions handled by Action classes. Still required.
- Keyboard and gamepad controls. Still required, but can be substituted with better controls.
- Controls based on elapsed time rather than fixed amounts. Still required.
- Display world axes. Not required, but useful for testing – recommended as an option, as well as a capability for disabling them.
- Detecting and handling collisions. If utilized in your game, can handle via the physics engine.
- Package organization. Still required except that starter package/class can be given a better name.
- Ability to run via command-line or batch file. Still required.
- Camera controller. Still required.
- Split-screen mode. Not required.
- Ground Plane. Not strictly required, but there must be terrain (and terrain-following) somewhere in your world.
- Scene-node controllers. Not required, but recommended. You may use the RAGE controllers.
- Hierarchical scene node(s). Still required – must include some hierarchical structure.
- Player's Guide. Still required as part of final report.
- Avatar. Still required – additionally, ghost avatar(s) are also required.

APPENDIX B

Grading breakdown:

Milestone 1	5
Milestone 2	5
Milestone 3	5
Game Day demo	5
Final Submission	20
total:	40 points