Project 3: Synchronized Networking

EGP405: Networking for Online Games (Fall 2017)

Assigned: **Monday, Oct. 30**

Due date: **Monday, Nov. 13**

Grade weight: **10%, out of 20 points**

Introduction:

In the last project, we explored the data coupled model and learned that fully synchronized peer states come at the cost of performance. Deliberately waiting for participants' data will stall the application and introduce severe lag. In this project we will implement a synchronized server-client model for a networked game or demo, in which participants update their own states in real-time.

Goals:

The goals of this project are:

1. Implement a local and networked mini-game or demo using RakNet.
2. Use timers and other techniques to synchronize remote game states.
3. Employ state prediction techniques to reduce dependency on data transfer.

Instructions:

Complete the following requirements:

1. *Local multiplayer prototype*: Implement a 2D *agent-based mini-game or demo* using an existing framework, or start with the boids demo from the last project. Be able to start a *local game* with unique data sets and/or controls for each player or participant. All players' data is updated in real-time and affects all other players. To make the next steps easier, make sure that each player's data is completely decoupled from all the others.
2. *Unsynchronized networking*: Implement the networked mode for your game by building a *server-client* architecture:
    a. The *server* is responsible for collecting updates from clients and routing messages; the server does not have a render window. The server should update the "main" game state on a time interval based on all current information and does not wait for client updates to do so.
    b. The *client* is responsible for connecting to the server and displaying the game state based on updates received from the server. Each client simulates its own agents and displays all agents from other clients given the most recent server update. Clients do not wait from updates from the server to update their own state.

    This can be done by building separate applications for the server and client, or having the ability to choose which one to act as when starting the game.

3. *Synchronized networking*: Use the event manager from the previous lab to help synchronize game states on all clients. Also do the following:
    a. Add *time stamping* to all real-time update messages.
    b. Receiving clients should determine the delay from the sender and calculate new offsets instead of automatically applying the update state.
4. *State prediction*: Extrapolate the current state based on previous updates.

Additional Requirements:
Complete the project in teams of ***up to three members*** with only one submission required per team. You must complete an independent peer review (see Canvas assignment) to receive a grade for your contributions. Be sure to ***test*** your project on multiple computers simultaneously before submitting.
Note: Expectations are proportional to team size!

Your team must also submit a ***brief justification document*** (1 – 2 pages) that outlines the system design. Provide a UML diagram of data structures used, and a diagram of the system's architecture (i.e. logical flow). This is not an essay or full design document; its purpose is to help me navigate your code. Provide team member contributions and point-form descriptions of where to find your features in the code.

To achieve a perfect score on this project, consider incorporating ***delighters*** in your system design. Implement at least one complementary feature not stated as a core requirement as above, and justify it in terms of the system design. The project is inherently creative, but this is an opportunity to do more with the project. Add things that are fun and unique.

Submission Guidelines:
1) Include the following header information at the top of all source files ***modified***:
    a. "This file was modified by <*names*> with permission from author."
2) Include the following header information at the top of all source files ***created***:
    a. Team member names and student IDs.
    b. Course code, section, project name and date.
    c. Certificate of Authenticity (standard practice): "*We certify that this work is entirely our own. The assessor of this project may reproduce this project and provide copies to other academic staff, and/or communicate a copy of this project to a plagiarism-checking service, which may retain a copy of the project on its database*."
3) All sections of code modified or written by your team should be commented, explicitly stating ***who*** made the change or wrote code and ***why***, and what the code ***does*** in the context of the program.

4) ***When you are finished***, delete all garbage files. ***Do not*** delete Visual Studio project or solution files, or your source files. ***Do not*** submit developer SDKs with your project (I gave them to you; I already have them). **Delete the following files**:
    a. ***All*** build and intermediate files (Debug, Release).
    b. Intellisense database files (including but not limited to *.db, *.sdf).
5) Zip up your project's root and rename using this convention: "**EGP-405-F2017-Net-Project3-<insert submitter's name here>**".
6) In the Canvas submission, add:
    a. Your design justification document as a PDF file.
    b. The above-mentioned zip file.
    Note: If the file size is larger than 1MB, you have not cleaned it correctly!
    c. A link to your public repository.
    Note: Grades will not be discussed or released through public repositories, for any reason, in compliance with FERPA.

Grading:
Grading for this project is broken into categories, with specific objective expectations listed in the table/rubric below.

Points **Category**	**0** **N/A**	**1**	**2** **Fair**	**3**	**4** **Meets Expectations**	**5** **Exceeds Expectations**
Functionality	Application does not serve the purpose or goals of the project.		Application serves its purpose with some bugs or issues.		Application serves its purpose with no bugs.	Application serves its purpose with "contingency plans" for user errors.
Features	Several required features not implemented.		Some of the required features implemented.		Most of the core features implemented; one or more delighters.	All required features and multiple delighters implemented.
Architecture	Code is not coherent, organized or modular.		Code has some organization and modularity.		Code is clean, coherent and organized into functions and modules where applicable.	Project makes use of appropriate file structure with full modularity.
Design	Design justification not provided.		Diagrams and/or descriptions provided but do not fully supplement the system implemented.		Diagrams and descriptions provided; adequately summarize features.	Diagrams and descriptions adequately summarize features, and help with navigating the code.

Specific penalties:

- **–10 points**: Submission does not build and run *correctly* and *immediately* out of the box (test using both debug and release modes).
- **–10 points**: Code not documented and commented, including contributions, explanations, certificate of authenticity, etc.
- **–10 points**: Lacking or weak evidence of having used version control to maintain project. Please ask for help if needed.
- **–10 points**: Submission contains junk files. Follow instructions above to ensure you have removed the correct files, and ask for help if needed.

Good luck, learn lots and have fun! ☺