# ***Lab 3: Networked Event Management***

***EGP405: Networking for Online Games (Fall 2017)***

Assigned: **Monday, Oct. 16**

Due date: **Monday, Oct. 30**

Grade weight: **5%, out of 10 points**

***Introduction***:

In recent lessons, we have explored some general networked game architecture principles that will help us build a synchronous networked game. We discussed multithreading and event management to help us determine the synchronous state of parallel activities.

***Goals & Result***:

This lab has two core requirements:

1. Implement a general event management system.
2. Use the system to dispatch networked game events.

***Instructions***:

Implement the following features in a console- or window-based networked application, either from scratch or based on one of the previous activities:

1. Event management:
    a. Implement a base class called *Event* that has a pure virtual function '*execute*' or '*dispatch*'.
    b. Implement an *event manager* class that has a ***non-STL queue/managed array*** of event pointers. The manager has two functionalities: add an event pointer to the queue, and execute all events in the order they were added. The manager can use the *singleton* design pattern for easy global access, or be passed in to any function that needs access to an event manager.
2. Networked events:
    a. Implement a networked application that has basic connection, real-time (synchronous) avatar movement and a few other basic player actions. How you show the gameplay is up to you.
    b. Implement at least ***five*** unique types of events that inherit from the base event class and have their execute function implemented. Each one should have a constructor that takes in data and stores the data as private members in the class. Each event should encapsulate some sort of unique gameplay-related purpose (does not include printing to the console). Unique means that the execution of each event type is vastly different from the others. Two events that ultimately have the same goal are not unique. Use member variables creatively to change the behavior of an event.

***Additional Requirements***:
Complete the lab in ***pairs*** and use version control often. Either set up a new project in a new repository (recommended) or use an existing repository that is unique to this team. See the briefing for lab 1 on instructions on setting up a new project in Visual Studio.

***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-Lab3-<insert submitter's name here>**".
6) In the Canvas submission, add:
    a. The above zip file.
    ***Note***: If the file size is larger than 1MB, you have not cleaned it correctly!
    b. 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***:

- **10 points**: complete and test all steps and sub-steps in the instructions.

***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!*** ☺