Term Project

Start Assignment

Due Jun 4 by 11:59pm **Points** 90 **Submitting** a file upload
File Types zip and tgz

Submit your code and writeup here.

Your term project is very open-ended. You will have to apply your knowledge from this class to make decisions as to how to exactly implement the project. Some key facts about this project:

1) You will be implementing a multiplayer game. This can be any multiplayer game. It can be text based or it can be graphical. It can be real-time or turn-based. It can be a well known game (like poker) or a game completely of your own creation. I recommend getting the basics of the protocol working and then working on building a fun game on top of it since you will be graded primarily on the protocol.

2) You will have to design the game networking protocol yourself. I will give you some protocol requirements, but anything beyond those is up to you. The network protocol will be encapsulated separate from your gaming code (eg, a class or set of classes). You will build an API so that the gaming code doesn't have to directly use sockets.

3) Deliverables will include the game, documentation of how to play the game and how the networking protocol works.

4) You may work in small groups of up to 3 people. If you do work in groups, you will have an additional requirement of documenting how you distributed the work, coordinated your development, handled any problems that arose. I will not handle any disputes between work partners or give extensions if one member of your group decides not to do any work or drops the class at the last minute, so choose your team wisely.

5) Each person or team will provide a 5 minute demonstration of their game in class and give the other students a chance to briefly play the game or see the game played. It is HIGHLY encouraged that you make your client executable available so that all the students can play your game. I will help facilitate this process. The students will then vote for the game they think is best (they cannot vote for their own game) and the team with the best game will get a 5 point bonus on their grade. I reserve the right to award multiple bonuses if several teams get a similar number of votes. If you need a system to set up your server, I have a Ubuntu machine available that I can host the server on, but contact me early so we can make sure it works.

6) There is no specified language. You can use C++, Rust, Java, Python, etc. If you want to learn a new language, this could be a good opportunity to do that.

7) You must use software version control. Use Github with your .edu address and mark the repository as private. Add me (BrentLagesse or lagesse@uw.edu (mailto:lagesse@uw.ed)) as an admin.

Assignment Details:

Your game may be in any network architecture model, including client/server, P2P, or hybrid, but if you must either support a game server that facilitates the game or a local broadcast for game discovery (the purpose of this is to avoid having to manually figure out people's IP addresses to start games). It must be written using sockets for network communication (no higher level networking constructs unless you build them yourself). Your game protocol will be written at the Application layer.

Your game must define and implement the following high level network protocol. You can add additional features to this if you want (and depending on your game, you might have to). This is a very common set of functions required by network games. If you decide to implement a very innovative game that inherently cannot support these functions, talk to me ASAP and we can discuss an alternative protocol:

1) Register

Registration announces that you are available to play the game and includes any contact information that is necessary for a player to establish a game with another player. This could optionally include a username/password to log into the system, but if you do include this information, do not send the password in plaintext. This is optional as part of the API if your game does not need to support it.

2) List Games

This provides a list of possible games that the user can join. The return value should include any necessary information that you might want the user to know about the games (for example, which level the game is playing or who is in the game). Typical approaches are to query the game server or the broadcast a game message to everybody on the LAN.

3) Create Game

This allows the user to announce that he/she has started a game and other people are allowed to join. This message should include any information that you want the user to know about the game such as what level the game is playing or who is in the game.

4) Join Game

This allows the user to join a game. The message should include any information necessary for the user to participate in the game.

5) Exit Game

This allows the user to exit the game gracefully.

6) Unregister

This should remove the user from the list of possible players, if you're maintaining one.

7) Application Specific protocol

You will have to design your own protocol for transmitting data between the players in the game (for example, showing your opponent that your tank has moved to a different position in a battle game or showing your opponent that you raised the bet in poker)

Bonus Features (5 points each)

8) Implement chat. This would enable users to chat with each other before games and in games.

9) Implement a scoreboard. This would keep a record of who has the most points or wins (or whatever your metric for scoring is). If you implement your game as a P2P games, you should use your bootstrap server as scoreboard server (a distributed scoreboard would be great, but it is beyond the scope of the course, so unless you're really motivated to do so, you can take the more centralized and simpler approach). The scoreboard should be persistent across all games (you can back it up to a file regularly on the server so if your server crashes, it can automatically reload it from the disk the next time it restarts).

Deliverables

In a zip file, you should turn in the following items:

1) Code for the game **(60%)**

2) Instructions for compiling and running the code. If you write a script to install dependencies, build, and execute your game, then just say to run that, this is basically just a free 5 points. **(5%)**

3) Documentation stating how you implemented each element of the protocol described above and what additions you made beyond the protocol described above. This should be well organized and easy to understand. **(30%)**

4) Documentation about how to play the game from logging on all the way to logging off. **(5%)**