[CS 260] Computer Networks I: Inter-process Communication

Programming assignment 4

1. Purpose of the assignment

The purpose of the assignment is to help the students do the following:

1. Demonstrate ability to write networking code through a simple LAN game running over UDP.
2. Demonstrate ability to implement basic networking game-related techniques: a dead-reckoning algorithm, and a lock-step protocol.

2. Specification

2.1. Requirements

This assignment can be implemented by individual students or self-arranged groups of 2-3 students; the larger the team, the better quality or complexity should be demonstrated in the submission. Each team member **must** be tasked with some part of networking code.

Your task is to implement from scratch a simple, fully playable real-time multiplayer game for **more than 2 players** using Winsock. The game must be implemented in C++ with the visual user┐interface done with a technology of your choice (i.e. OpenGL, Windows GDI, SDL, 3rd-party frameworks are allowed if they are free for commercial use).

Some ideas for games you can use:

- Cooperative asteroids for 3-4 players (each player controls a separate spaceship, while asteroids fly through a map),
- Competitive pong for 3-4 players (each player controls a paddle at one side of a map, bounding a ball),
- Cooperative real-time dungeon crawler.

Your team is given creative freedom, but keep the game mechanics **very simple**, and focus on the networking aspects of the code. The program must be a real-time game; turn-based games are not permitted. The concept, style, entertainment factors and implementation of elements other than the game loop and networking will not be considered during grading, unless they demonstrate the quality deserving some discretionary bonus marks.

All the network communication must be done using the **peer-to-peer** architecture via your own **UDP**-based protocol. There should be no dedicated server program; each client should run one and the same executable.

To launch a new game the user should be able to specify the following command-line parameters:

```
>game.exe hostname1:port1 hostname2:port2 hostname3:port3 hostname4:port4
```

Each *hostname* parameter should be an IP address or a host name of a host where a client is (or will be) running, while the corresponding *port* should indicate a UDP port at which that client is expected to listen. Take note that a copy of a game client can be started when other clients have not been started yet. A game should begin only when all clients are available online.

The game must include the following elements:

- Playable agents, each controlled by an individual human player connected via network (i.e. in pong each player controls one paddle).
- Non-playable agents in motion that players can interact with (i.e. in pong the ball is constantly traveling and can be hit with a paddle). Such agents may move on a simple linear trajectory.
- Implementation of the dead-reckoning algorithm (to be covered in a lecture).
    - As the game will include playable agents and non-playable agents in motion; once put in motion, all network clients will have to **predict** per frame their future positions and do it simultaneously and consistently, without exchanging information. They also have to correct it in cases when the position has been predicted inaccurately compared to an occasional update packet or an agent-related event.
    - Clients should synchronize creation, destruction of passive agents, and synchronize gameplay related input, i.e. steering.
    - Optionally, occasionally (but definitely not per frame!) clients can synchronize game state representing current behaviour of agents.
    - Movement of agents must be smooth and consistent for all clients.
- Implementation of a lock-step protocol (to be covered in a lecture).
    - Clients must announce that they have committed to take a certain action (by sending a *hash value* of the action to other clients).
    - Once a client receives such announcement from all other clients, only then it announces the details of the action, which must match the hash.
    - All clients validate the hash, and if it matches recognize the action taken.
    - You can use any hash algorithm of your choice; please encapsulate it in a separate header file or a class. You are allowed to use third party libraries on licenses permitting free commercial use, such as PicoSHA2. Alternatively, you can use an implementation of a CRC checksum to replace a hash.

While implementing the game consider the following assumptions:

- The team must design an Application Layer protocol most suitable for a selected game type. Plan the protocol carefully, consider utilizing broadcast where applicable (or everywhere if suitable).
- The game must run at a *localhost* and over *LAN*, but not via the Internet (due to UDP broadcasts).
- No packets are lost - for simplicity we assume that all packets arrive uncorrupted but not necessarily in the right order. There is no need for implementing complex *reliable data transfer* features or pipelining.
- There is a need to reduce the network traffic. Both the size of packets and their frequency should be optimized for this purpose..

Additionally, a submission must include a single 1 page PDF document (longer documents will require resubmission!), with:

- A list of team members and their key responsibilities in the area of networking.
- A brief summary highlighting how the network communication is handled by the game (what data is exchanged, when, how does this approach reduce network traffic); a simple high-level bullet-point list is sufficient.

2.2. Deliverables

Submissions will be graded manually. Each submission must be provided as a ZIP file with the structure matching the one presented below, uploaded using the Moodle submission link. All members of the same team should upload a copy of the same submission as their own submission independently; the instructor will grade the work using any of the copies.

To prepare the ZIP file perform the following steps:

a) Inside a folder put a single, complete Visual Studio 2019 solution with all required C++17 projects. Include the PDF document as *readme.pdf* in the same folder as the **.sln* file.

b) Compress the files (**without including the root folder**) in a ZIP archive.

c) Rename the ZIP archive to *foo.bar_cs260_4.zip* (replace *foo.bar* with your own login).

Upload the ZIP package using the Moodle submission link.

No other files must be present in the ZIP package. **Do not include unnecessary binary files or the compilation outputs in the package**.

The C++ code must compile in the *release mode* (not *debug mode*) in Visual Studio 2019 (or newer) without any warnings and run without memory leaks to receive a non-zero grade. To prevent memory leaks, remember to release all resources!

Make sure your name and other relevant information is properly filled-in in header comments of source files. Apply a proper programming style that you have practiced in CS 170 and CS 225.

2.3. Rubrics

Submissions must be uploaded to the *DigiPen Distance Learning* course management system (Moodle), and they are assessed in the following way:

- Critical penalties
  - Plagiarism. Additional consequences will follow.
  - Submission has not been uploaded within a required deadline. Do not wait until the last minute, as uploading the work after the deadline may be impossible.
  - Submission does not build properly (a project does not compile). Test your code.
- Major penalties
  - Each essential memory leak. In our assignments, **resource management is king**.
  - Significant noncompliance with the specification, even if the output is correct. Read the specification carefully.
- Minor penalties:
  - It must be possible to start a game with all clients online.
  - A player (or all cooperating players; depending on a type of a game) must be able to win and to lose a game.
  - There must be non-playable agents to interact with that can move independently.
  - Game must implement dead-reckoning algorithm at least for non-playable agents.
  - Game must implement lock-step protocol and use a hashing algorithm.
  - All clients must gracefully terminate when one of clients is terminated.
  - Inconsistent file names, coding style, commenting and general shortcomings in code quality, even if the code works.
  - Minor violation of the specification.

The list above is non-exhaustive. The lecturer reserves the right to impose reasonable penalties for code that violates general practices or does not match the specification in an obvious way that has not been mentioned above. In exceptional cases, the lecturer reserves a discretionary right to allow resubmission or submission after the deadline.

Bonus marks

Extra features that are relevant to the assignment are welcome, but all the required features must be implemented reasonably well first. Extra features must be **highlighted** in the PDF document.

If the game supports the ability to reconnect players to an ongoing game and continue after *multiple* (up to all but one) clients got unexpectedly terminated, the bonus marks will be no more than **3% of the course total grade**; otherwise no more than **20% of the assignment 4 grade** (capped at 100%).