Project Milestone 9

CS 4500 Software Development

Due: ~~Wednesday, April 14~~Friday, April 16, 11:59pm

Submission: Create a release on GitHub tagged `p9`. Add a `p9-exec.zip` to the release, containing:

1. The `snarlServer` and `snarlClient` executables for the testing task
2. A `README.md` file explaining how to start the server, and how to run and interact with the client.
3. Any packages/modules necessary for the above executables to function

Here is a sketch of the `p9-exec.zip` layout:

```
|-- README.md
|-- snarlClient
|-- snarlServer
...
```

Submit a ZIP with the following on Handins:

1. For the programming task, the server and client components source files under `Snarl/src/Remote/`, as well as any relevant updated source files under `Snarl/src/`
2. For the testing task, a folder `Snarl/net/` with the source code for `snarlClient` and `snarlServer`.

Here is a sketch of an example Handins ZIP layout:

```
Snarl
|-- net
|   |-- README.md
|   |-- snarlClient.<ext>
|   |-- snarlServer.<ext>
|   ...
`-- src
    |-- Remote
    |   |-- Client.<ext>
    |   |-- Server.<ext>
    |   ...
    ...
```

The objective of this Milestone is to implement the networked client-server architecture of SNARL. At this point in the project, your code base contains almost all the components needed to accomplish this. The remaining work is to write "wrappers" that enable these components to talk to each other over TCP sockets. From an OOD perspective, these wrappers might be instances of design patterns known as *Remote Proxy* (which will be introduced in a lecture/video) and *Adapter* (which you have seen in CS3500).

Programming Task

Implement a remote player *client* and a *server* component for SNARL.

The client component's purpose is to connect the player front-end to the server via TCP, and allow playing a game, following the SNARL protocol specification.

The server waits for player clients to connect via a TCP socket. After each player connects, the server should wait for up to a given number of seconds (60 by default) for the next player, until the maximum number of players is reached. The maximum number can be a server instance-specific maximum number (e.g., supplied via command line) or 4, whichever is lower.

Once at least one player is ready to play (and timeouts have lapsed), the first level begins. When the game is over (all levels have been completed, or due to another reason), the server closes down all connections and shuts down.

Testing Task

The goal of this "testing" task is to provide two executables: `snarlServer` and `snarlClient`.

snarlServer The `snarlServer` executable starts your server. It should take the following optional command line arguments:

- `--levels FILE`, where `FILE` is the path and name of a file containing a JSON level specifications as described in Milestone 8. Default is `snarl.levels` (in the current directory).
- `--clients N`, where $1 \leq$ `N` ≤ 4 is the maximum number of clients the server should wait for before starting the game. This option determines `max_clients` in the protocol specification. Default is `4`.
- `--wait N`, where `N` is the number of seconds to wait for the next client to connect. This option determines `reg_timeout`. Default is `60`.
- `--observe` – when this option is given, the server should start a local observer to display the progress of the game.
- `--address IP`, where `IP` is an IP address on which the server should listen for connections. Default is `127.0.0.1`. You can choose to support a hostname (e.g., `localhost`), but this is not required.
- `--port NUM`, where `NUM` is the port number the server will listen on. Default is `45678`

After starting, the server should wait for clients to connect on the specified address and port. Once the maximum number of clients has connected or the timeout has lapsed, the server should run a game according to the SNARL protocol specification and the local game description given in Milestone 8.

snarlClient The `snarlClient` executable starts your player client. It should take the following optional command line arguments:

- `--address IP`, where `IP` is an IP address the client should connect to. Default is `127.0.0.1`. You can choose to support a hostname (e.g., `localhost`), but this is not required.
- `--port NUM`, where `NUM` is the port number the client should connect to. Default is `45678`

As soon as `snarlClient` starts up, it should prompt the user for their name, and attempt to connect to the server on the given address and port. After connecting, it should behave according to the protocol and allow the user to play a SNARL game on the server.

README.md Write up a `README.md` explaining how to use `snarlServer` to run and `snarlClient` to connect to networked SNARL games. The description for the client should include basic instructions on how to play the game. The executables need to be runnable and playable on Khoury Linux VMs, via `ssh` or using the Khoury Virtual Desktop. Include information about whether the client or the server's observer can be used in a terminal or they require an X session to display a GUI.

Expectations The client and server have to adhere to the protocol, allowing us to connect another team's (or our) client and use it with your server, or vice versa. The game should progress similarly to `localSnarl` from Milestone 8, modulo differences introduced by the protocol or the present milestone description.