Game Servers

David Hoogenbosch
eingt2a
304734

Protocol Reflection

To get commands across the network I wrap them in classes that can send using SFML packets. This allows for easy use of structures to pass data from the client to the server and vice versa.

Sending Messages:

I use a form of remote procedure calls to inform the server and clients about events that have occurred. Commands are in classes, classes are wrapped in sfml packets. These packages are sent to the other side. A thread on either end of the connection is constantly waiting for new messages. When it receives a message it is put in the incoming message queue. This queue preserves the order that they have arrived in. Every update loop the client and the server empty this queue (using Mutexes). These messages are sent to a class that can reconstruct the original classes. A specialization of this class handles the differentiation between Game and Server.

Keeping state:

The server is authoritative The server knows what the clients want to do because they send keystate updates(only when it changes). The server uses this input and extrapolates the position of the client by simulating the time that has passed between the updates. Clients can predict where they should be based on the last received input from the server the same way. A client's character inherits from the same class that the server uses for simulation.

At the end of every loop on the server (0.2 seconds by default) the server sends the new gamestate over udp. This new state contains the accurate position of the players in the world and their last known keyboard state. I personally feel that this could be improved by replacing it with a tcp message only when it is detected that a player has changed its keyboard input.

Because udp connections get blocked by firewalls the gamestate updates are currently not arriving when used over the internet. They work fine when used over localhost.

The login sequence:

A player enters his login data in the client. He enters the ip of the server he wants to join. He then chooses between login or register.

A message is sent over to the server containing the players information.

If they player selected login, the server will check if there is a match between the username and the password. If there is a match the player receives a set of tcp messages to bring him up to date with the current state of the server.

If the player chooses register, the server will check if the username is available. If the username is available a new entry is made in the database for the user. The server sends a rejection event to the client with a message saying “Created user in database, you can now login”. The user logs in with his new account and the game starts.

Kicking off:

To start a game the client must first be informed about the state of the server.
This is done by a series of tcp messages.

1: Connected players
For each connected player on the server the new client receives one player connection event. When this message is received by the client, the client will create new characters locally. These characters are linked to the client manager.

2: Loaded level
Server sends one load level command to make the client load a specific level.
3: State of the targets
Targets are pickups in the game that reward score. A list of ids and a series of positions is sent by the server. The client places visual markers on these positions.
4: Score
Server creates sends a score update to the client so that he has the latest score.

Disconnection:
When a player closes their sfml window, the game gets cleaned up. The only thing that remains for a brief moment is the clients networkinterface class. The network interface sends a disconnection event to the server before it shuts down. This ensures that the server knows what players are active. The server can constantly remain running and receive new connections.

Extras:

There are various options for the game in the config.json file. This allows for automation of the login for testing purposes. A master server can set up and queried for active servers. Currently

the information about active servers is only displayed in the clients console (this requires some port forwarding).

Sql Queries are loaded from config.json to make for easier editing. Background textures are generated in game based on a comma separated list in the config file.