Network Programming Coursework

Alberto Taiuti - 1300250

Network Architecture

- Client – Server Architecture
  - Easier to deal with authority of objects
  - Performance does not rely on all clients having a good internet connection
  - Simpler to develop, fits better the scope of the project
  - Avoids NAT problems so it can be easily expanded to non local networks

Network Architecture

- TCP used as protocol
  - Simpler to use than UDP
  - Reliable and generally performs good on nowadays networks
  - Good fit for the scope of the application
  - Nagle disabled to overcome some limitations of TCP with respect to low latency

Network Architecture

- Asynchronous Sockets
  - Reduces direct CPU usage
  - Simplifies API, making it easier to use and more straightforward
  - Less code required from the user of the library thus reducing possible errors

  - Used Microsoft’s implementation

Software Architecture

- Object Oriented Design
  - Game classes
    - Keep code separated from the main
    - The source is more readable and organised
    - Handle gameplay and high-level networking
  - GameState classes
    - Further organise the source
    - Easier to use and to maintain
    - Create a state machine which changes based on the state of the game
    - Execute state-based functionality

Software Architecture

- Network classes
  - Hide networking complexity from main game classes
  - Wrap the C-style API
  - Expose network functionality to other classes
- Entity classes
  - Represent entity in the world

State classes

- Event handlers
  - Different game states handle events differently
  - Use polymorphism to adapt child classes
- Message parsing
  - Read queue of messages
  - Perform appropriate action based on message and other factors
- Other main functions
  - Update, Render, etc.

Application level protocol

- NetMessage base class, holds:
  - ID of client
  - Type of message
  - Data
- Data is a union of structs
- Each struct is for a different type of message
  - This way, there's only one type of message which is exchanged

Application level protocol

- After two clients connect, the game starts
- Client sends Input messages containing
  - Snapshot of input
  - Input sequence number used for reconciliation
  - ID
- Server processes messages
  - Using simulated network delay
  - All the queued ones in one frame

Application level protocol

- Server holds master copy of world
- Sends periodical world snapshots to clients, containing
  - Position of entity
  - Velocity of entity
  - Direction of entity
- … for all the entities in the world

Low-level networking

- The network classes hide complexity
- Their interface accepts NetMessage objects

- Socket connection is represented by the Connection class
- Connection class handles socket lifetime

- Network classes use one or a collection of Connection objects to manage connections
  - Server has a map, client's only one

Low-level networking

- Network classes are adapted per-application
  - Client's is different from the server's
  - Server's provides method for propagating a message to all the connected clients
- Connection class further hides complexity of API
- Provides easy way of interaction in an object oriented fashion
- When destroyed, they clean-up the socket connection

Interpolation

- Client sees other entities in the past
- The other entities are shown by interpolating between the two last received position updates for the entity

- When position update arrives (for entities which are not the one controlled by the client)
  - Update last and previous position
  - Calculate velocity out of them
  - Assign it as the velocity of the entity
  - Use the normal update method on the entity

Interpolation

- This way, no need for special interpolation method!
- Bullets' interpolation work exactly the same as tanks by means of polymorphism
- This method was described by Valve and Gabriel Gambetta:
  - https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking
  - http://www.gabrielgambetta.com/fpm3.html

Interpolation

- Chosen interpolation with data from the past rather than interpolation for correction
- The tanks can make sharp turns
- It is more effective to show the other entities in the past than constantly correcting from the prediction (for other entities not under control of the client)

Prediction

- When activated, entity controlled by client directly applies input messages which sends to server, immediately
  - Set velocity
  - In update, use that velocity to move independently from the server authority
- .. But when the server's authoritative message arrives, entity bounces back!

Reconciliation

- Hence the need for reconciliation
- Reconciliation can be activated or deactivated
- When active, uses input sequence numbers
  - If server hasn't acknowledged a certain input
    - Re-apply it
- Store queue of sent messages which haven't been acknowledged
- Using reconciliation the client's controlled entity appears moving smoothly, hence the decision to using it.

Project outcomes - Positives

- Classes and functions' responsibilities are nicely separated
- Use of state pattern simplify implementation of new states
- Learnt very much about network programming and increased the awareness about the subject in general
- Use of Reconciliation with Prediction, a topic extra to the lectures
- Implementation allows for a smooth experience even when lag (even if simulated) is present and the server ticks at a lower rate than the clients

Project Outcomes - Negatives

- Use of TCP could lead to large delays on a real network due to packet loss and related retransmission
- Server always sends data, does not use delta compression – much data is redundant
- Lag compensation is not implemented for bullets shooting and bullets appear in the past: non-optimal on high-delay networks
- The shooting button, J, is fiddly

Possible improvements

- Use UDP instead of TCP, and develop an application-layer protocol based on it this avoiding TCP's retransmission
- Use delta compression for server position snapshots
- Fix the shooting button J
- Have tweakable latency, game tick, etc. parameters

Thanks for listening!