Module 4

Networking
Networking

STRATEGIES
The Illusion

- All players are playing in real-time on the same machine
- This isn’t possible
- We need to emulate this as much as possible
The Illusion

- What the player should see:
  - Consistent game state
  - Responsive controls
  - Difficult to cheat

- Potential problems
  - Game state > bandwidth
  - Variable or high latency
  - Antagonistic users
Send the Entire World!

- Players take turns modifying the game world and pass it back and forth
- Works alright for turn-based games
- Usually this is bad
  - RTS: millions of units
  - FPR: millions of bullets
  - Fighter: timing is crucial
Modeling the World

● If we send everything we model the world as if everything is equally important
  ○ This isn’t necessarily true
  ○ Not all entities react to the player
  ○ The player doesn’t need every level if they are only on level 1

● We need a better model to solve these problems
Client-Server Model

- One process is the authoritative server
  - Now we don’t have to wait for slow players, just the server
  - Server can either be hosted by a player or on a separate computer
- Other processes are “dumb terminals”
  - Sends all input to server
  - Server updates the world and sends it back
- Problem: client has to wait for server to respond to perform even basic actions
Client-side Prediction

- Client responds to player input immediately
- When the server sends back the authoritative game state, client state is overwritten
Rollback

- What if the server sends a game state that was 100ms in the past?
- We can’t just replace our game world because we lose all the commands from the local player
  - Client has to roll back the world and integrate commands since the last known good state
Masking the Timewarp

- Laggy players experience this jump often
- If the server usually sends states from 100ms ago, run the client 100ms behind
- Turns a jumpy experience into a smooth and only slightly slow one
  - Very useful if relative timing of commands is important
What about the server?

- Without rollback and with masking:
  - In an FPS, would need to lead shots because the game state displayed is in the past
- With rollback and without masking:
  - The target could be shot after they think they have taken cover (by a less laggy player)
  - Or we could delay the server player as well
- Need to think carefully about both technical requirements and game impacts of any networking model
QUESTIONS?
Networking

IMPLEMENTATION
TCP: Transmission Control Protocol

- Abstracts over IP
- All packets are guaranteed to be received and in the correct order
- Good for sending important, permanent data
  - Connecting to server
  - Initial game state
UDP: User Datagram Protocol

- A very thin shell around IP
- Much faster than TCP, but no guarantees about reception or order
- Good for information where only the most recent state matters
  - Current game state

![UDP diagram](image-url)
TCP vs UDP

- (Very) generally: action games use UDP and turn-based games use TCP
  - World state updates can be lost without worry, commands not so much
- Can potentially combine them
  - TCP sends one-time messages
  - UDP streams updates
- Best choice varies by project
Java Sockets

- Very good for most purposes
- Read and write objects to sockets
- UDP is deprecated for sockets
  - Use DatagramSocket
Setting up Sockets

- Open a connection on a port
- Open an input/output stream from the socket
- Read and write to the streams (which use the socket’s protocol)
- Close the streams and sockets

```java
String host = "127.0.0.1";
int port = 10800;
Socket out = new Socket(ip, port);
ObjectOutputStream stream;
stream = new ObjectOutputStream(out.getOutputStream());
stream.writeObject("HelloWorld");
stream.close();
out.close();

String host = "127.0.0.1";
int port = 10800;
Socket in = new Socket(ip, port);
ObjectInputStream stream;
stream = new ObjectInputStream(in.getInputStream());
System.out.println(stream.readObject());
stream.close();
in.close();
```
Edge Cases

● What if...
  ○ The client disconnects
  ○ The server dies
  ○ The client goes insane and send gibberish
  ○ The client loses internet for 30 seconds
  ○ The client is malicious
  ○ The client changes IP address

● Handling errors well is vital to the player experience
Elegant Disconnects

- Handles and respond to IO exceptions
  - Don’t just dump a stack trace
- Display informative status messages
- Send heartbeat packets every few seconds
  - Then respond if server / client hasn’t received a heartbeat in a while
- Never let the game continue to run in an unrecoverable state!
QUESTIONS?