Lecture 11

Networking
Consistency

- Do our games agree?
  - Where do I see objects?
  - Where do you see them?
  - Who is authoritative?
- How to force agreement?
  - Do I wait for everyone?
  - Do I guess and fix errors?

Security

- What cheats are possible?
  - View hidden data
  - Enter invalid states
  - Improve player skill
- How do we cheat proof?
  - Technical solutions?
  - Community policing?
CS 3152: Game Networking Issues

Consistency

- Do our games agree?
  - Where do I see objects?
  - Where do you see them?
- How to force agreement?
  - Do I wait for everyone?
  - Do I guess and fix errors?

Security

- What cheats are possible?
  - View hidden data
  - Enter invalid
- How do we cheat proof?
  - Technical solutions?
  - Community policing?

Today’s Lecture

Not going to cover
The Issue of Consistency

- **Latency** is root of all evil
- **Local** actions are instant
- **Network** actions are slow

**Example:** targeting
- Want “geometric fidelity”
- Fire a weapon along ray
- Hits first object on ray
- But movement is fast!

How to tell these cases apart?
World State vs. Local State

- **State**: all objects in game
  - **Local State**: on a machine
  - **World State**: “true” state
- **Where** is the world state?
  - On a single machine?
  - Union of local states?
- States may be *inconsistent*
  - Local disagrees with world
  - Is this really a problem?
  - What can we do about it?
The Question of Authority

Centralized Authority

- One computer is authority
  - Stores the full world state
  - Local states must match it
- Often call this the “server”

Distributed Authority

- Authority is divided up
  - Each object has an owner
  - Must match if not owner
- Classically call this “P2P”
Authority and Latency

- Lack of authority enforces a delay
  - Only draw what authority tells you
  - Requires round trip from your input
  - Round-trip time (RTT) can be > 200 ms

- This makes the game less responsive
  - Need some way to compensate for this
**Authority and Latency**

- Lack of authority enforces a delay
  - Only draw what authority tells you
  - Requires round trip from your input
  - Round-trip time (RTT) can be > 200 ms

This makes the game less responsive
- Need some way to compensate for this

---

**Networking**

- Sample Input
- Render Screen
- Process Input
- Compute State

---

Need to understand basics before solving this
## Networking Breaks into Two Phases

<table>
<thead>
<tr>
<th>Matchmaking</th>
<th>Game Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Service to find other players</td>
<td></td>
</tr>
<tr>
<td>• Groups players in a session</td>
<td></td>
</tr>
<tr>
<td>• But does not run session</td>
<td>• Service to run the core game</td>
</tr>
<tr>
<td>• Why make your own?</td>
<td>• Synchronizes player state</td>
</tr>
<tr>
<td>• Control user accounts</td>
<td>• Supports minor adds/drops</td>
</tr>
<tr>
<td>• Implement skill ladders</td>
<td>• Why make your own?</td>
</tr>
<tr>
<td>• 3rd party services common</td>
<td>• Must tailor to your game</td>
</tr>
<tr>
<td>• Apple GameCenter</td>
<td>• You often have no choice</td>
</tr>
<tr>
<td>• GooglePlay API</td>
<td>• Limited 3rd party services</td>
</tr>
<tr>
<td>• Unity’s server classes</td>
<td>• Often just a networking API</td>
</tr>
<tr>
<td></td>
<td>• For limited class of games</td>
</tr>
<tr>
<td></td>
<td>• <strong>Examples</strong>: Unity, Unreal</td>
</tr>
</tbody>
</table>
## Networking Breaks into Two Phases

<table>
<thead>
<tr>
<th><strong>Matchmaking</strong></th>
<th><strong>Game Session</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Service to find other players</td>
<td>• Service to run the core game</td>
</tr>
<tr>
<td>• Groups players in a session</td>
<td>• Synchronizes player state</td>
</tr>
<tr>
<td>• But does not run session</td>
<td>• Supports minor adds/drops</td>
</tr>
<tr>
<td><strong>Simplify if possible</strong></td>
<td><strong>Our main focus</strong></td>
</tr>
<tr>
<td>• Why make your own?</td>
<td>• What else?</td>
</tr>
<tr>
<td>• Control user accounts</td>
<td>• Must tailor to your game</td>
</tr>
<tr>
<td>• Implement skill ladders</td>
<td>• You often have no choice</td>
</tr>
<tr>
<td>• 3rd party services common</td>
<td>• Limited 3rd party services</td>
</tr>
<tr>
<td>• Apple GameCenter</td>
<td>• Often just a networking API</td>
</tr>
<tr>
<td>• GooglePlay API</td>
<td>• For limited class of games</td>
</tr>
<tr>
<td>• Unity’s server classes</td>
<td>• <strong>Examples</strong>: Unity, Unreal</td>
</tr>
</tbody>
</table>
Matchmaking: Apple/iOS

- Uses the **GameKit** library
  - Supports multiplayer games
  - Also leaderboards/achievements
  - Not a full game engine

- Very simple matchmaking
  - Specify the number of players
  - Invite anyone on friends list
  - Invite anyone in BlueTooth range
  - Or allow Apple to hook you up

- Can be simultaneous with session
  - Add more players if slots available
iOS Matchmaking Classes

Real Time
- You handle authority
  - Allows variety of strategies
  - Focus of rest of lecture
- **GKMatchmakerViewController**
  - Classic matchmaking UI
  - You add a listener/delegate
- **GKMatchmaker**
  - Controller with no UI
  - Allows a custom view

Turn Based
- Apple handles authority
  - Stores state on Apple server
- **GKTurnBasedMatchmakerViewController**
  - Classic matchmaking UI
  - You add a listener/delegate
- **GKTurnBasedMatch**
  - Controller with no UI
  - Allows a custom view
## iOS Matchmaking Classes

### Real Time
- You handle authority
  - Allows variety of strategies
  - Focus of rest of lecture
- **GKMatchmakerViewController**
  - Classic matchmaking UI
  - You add a listener/delegate
- **GKMatchmaker**
  - Controller with no UI
  - Allows a custom view

### Turn Based
- Apple handles authority
  - Stores state on Apple server
- **GKTurnBasedMatchmakerViewController**
  - Classic matchmaking UI
  - You add a listener/delegate
- **GKTurnBasedMatch**
  - Controller with no UI
  - Allows a custom view

---

*Will require you to use Objective-C++*
Advantages of a Custom UI

Key if mixing AI and multiplayer
Matchmaking: Android

• Part of the Google Play API
  • Supports multiplayer games
  • Also leaderboards/achievements
  • Also some minor game analytics
• Works exactly like GameKit
  • Choose real-time or turn-based
  • Use Google UI or a custom one
  • Only differ in terminology
• Has a native C++ API
  • No need for Java or JNI
  • See reading for documentation
Custom Matchmaking

• Typically need to have a separate server
  • Fixed, hard-coded IP that your app connects to
  • Custom user accounts that you manage
  • How Unity works (though they give software)

• Commonly used for **Firebase** networking
  • One app creates a session with Firebase server
  • Other apps connect to session on ad-hoc basic

• **Benefit**: cross-platform matchmaking
  • Only way for iOS to play with Android
Matchmaking in *Family Style*

**KITCHEN CREATED!**

**SHARE YOUR KITCHEN AND CONTINUE ONCE EVERYONE HAS JOINED!**
Firebase Caveats

• Really only works if network traffic is low
  • Ideal for turn-based strategy games
  • Family Style only used it to “pass” food

• Virality can be expensive
  • Firebase is free only if traffic is low
  • Family Style got charged 2k overnight
  • Plan for the worst in your design

• Platform turn-based options are cheaper
Game Session: Part of Core Loop

Client

Update

Draw

Authority

Update

Networking
Decoupling the Network Loop

Client

- Local Update
- Draw

Network Update

Authority

Update

Networking
Decoupling the Network Loop

Client

- Local Update
- Draw

Authority

- Update

Network Update

Possibly slower tick rate (10 fps)

Smooth local animation

Should match the client rate

Networking
Decoupling Enables Latency Masking

- Animation is “buying time”
  - Looks fast and responsive
  - But no real change to state
  - Animation done at update

- **Examples:**
  - Players wait for elevator
  - Teleportation takes time
  - Many hits needed per kill
  - Bullets have flying time
  - Inertia limits movement
**Game Session: Dedicated Server**

- **Server developer provides**
  - Acts as central authority
  - May be several servers
  - May use cloud services

- **Pros:**
  - Could be real computer
  - More power/responsiveness
  - No player has advantage

- **Cons:**
  - Lag if players not nearby
  - Expensive to maintain
Game Session: AdHoc Server

- One client acts as host
  - Acts as central authority
  - Chosen by matchmaker
  - But may change in session

- Pros:
  - Cheap long-term solution
  - Can group clients spatially

- Cons:
  - Server is a mobile device
  - Host often has advantages
  - Must migrate if host is lost
**Game Session: AdHoc Server**

- One client acts as host
  - Acts as central authority
  - Chosen by matchmaker
  - But may change in session

- **Pros:**
  - Predominant commercial architecture

- **Cons:**
  - Server is a mobile device
  - Host often has advantages
  - Must migrate if host is lost
Game Session: True P2P

• Authority is distributed
  • Each client owns part of state
  • Special algorithms for conflict
  • Coordinator for adds/drops

• Pros:
  • No lag on owned objects
  • Lag limited to “attacks”
  • Same advantages as adhoc

• Cons:
  • Incredibly hard to implement
  • High networking bandwidth
**Game Session: True P2P**

- **Authority is distributed**
  - Each client owns part of state
  - Special algorithms for conflict
  - Coordinator for adds/drops

- **Pros:**
  - Almost no-one does this outside academia
  - No lag on owned objects
  - Lag limited to "attacks"
  - Same advantages as adhoc

- **Cons:**
  - Incredibly hard to implement
  - High networking bandwidth
Game Session: True P2P Networking
Game Session: True P2P

Melee is easy to latency mask!
• Clients must be **synchronized**
  • Ensure they have same state
  • … or differences do not matter

• Synchronization != authority
  • Authority determines true state
  • Not *how* clients updated
  • Or *when* clients are updated

• Major concept in networking
  • Lots of complicated algorithms
  • Also a **patent mindfield**
  • Take distributed systems course
## Synchronization Algorithms

<table>
<thead>
<tr>
<th>Pessimistic</th>
<th>Optimistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Everyone sees same world</td>
<td></td>
</tr>
<tr>
<td>- Ensure local = world state</td>
<td></td>
</tr>
<tr>
<td>- Forces a drawing delay</td>
<td></td>
</tr>
<tr>
<td>- Best on fast networks</td>
<td></td>
</tr>
<tr>
<td>- Local LAN play</td>
<td></td>
</tr>
<tr>
<td>- Bluetooth proximity</td>
<td></td>
</tr>
<tr>
<td>- Or games with limited input</td>
<td></td>
</tr>
<tr>
<td>- Real time strategy</td>
<td></td>
</tr>
<tr>
<td>- Simulation games</td>
<td></td>
</tr>
<tr>
<td>- Allow some world drift</td>
<td></td>
</tr>
<tr>
<td>- Best guess + roll back</td>
<td></td>
</tr>
<tr>
<td>- Fix mistakes if needed</td>
<td></td>
</tr>
<tr>
<td>- Works on any network</td>
<td></td>
</tr>
<tr>
<td>- Lag errors can be fixed</td>
<td></td>
</tr>
<tr>
<td>- But fixes may be distracting</td>
<td></td>
</tr>
<tr>
<td>- Works great for shooters</td>
<td></td>
</tr>
<tr>
<td>- Player controls only avatar</td>
<td></td>
</tr>
<tr>
<td>- All else approximated</td>
<td></td>
</tr>
</tbody>
</table>
# Synchronization Algorithms

<table>
<thead>
<tr>
<th>Pessimistic</th>
<th>Optimistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Everyone sees same world</td>
<td>Allow some world drift</td>
</tr>
<tr>
<td>Ensure local = world state</td>
<td>Best guess + roll back</td>
</tr>
<tr>
<td>Forces a drawing delay</td>
<td>Fix mistakes if needed</td>
</tr>
<tr>
<td>Best on fast networks</td>
<td>Works on any network</td>
</tr>
<tr>
<td>Local LAN play</td>
<td>Lag errors can be fixed</td>
</tr>
<tr>
<td>Bluetooth proximity</td>
<td>But fixes may be distracting</td>
</tr>
<tr>
<td>Or games with limited input</td>
<td>Works great for shooters</td>
</tr>
<tr>
<td>Real time strategy</td>
<td>All else approximated</td>
</tr>
<tr>
<td>Simulation games</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Also great for distributed authority</td>
</tr>
</tbody>
</table>

Networking
Pessimistic: Lock-Step Synchronization

- **Algorithm**: play by “turns”
  - Players send turn actions
  - Even if no action was taken
  - Wait for response to render
- **Problems**
  - *Long* Internet latency
  - Variable latencies (jitter)
  - Speed set by slowest player
  - What if moves are lost?
- More common in LAN days
**Algorithm**: turns w/ timeout
- Often timeout after 200 ms
- But can be adapted to RTT
- All moves are buffered
- Executed at end of next turn

**Problems**
- Variable latencies (> a turn)
- Speed set by slowest player
- What if moves are lost?

**Used in classic RTS games**
Pessimistic: Bucket Synchronization

- **Algorithm**: turns w/ timeout
  - Often timeout after 200 ms
  - But can be adapted to RTT
  - All moves are buffered
  - Executed at end of next turn

- **Problems**
  - Variable latencies (> a turn)
  - Speed set by slowest player
  - What if moves are lost?

- Used in classic RTS games
Optimistic: Personal State

- Local Update
  - Draw

- Network Update

- Server Update

Current Network State

- Unconfirmed player actions
  - Approx. Current State

Previous Network State

- State updates
  - Player action confirmations
  - True State

Networking
Optimistic: Opponent State

- Local Update
- Draw
- Network Update
- Server Update

Current Network State
- Simulate assuming no actions
- Approx. Current State

Previous Network State
- State updates
- Opponent actions

Current Network State
- Networking

True State
Advantages of Sending Actions

Dead Reckoning
- Assume velocity constant
  - Simulate the new position
  - Treats like physics object
- Generalize to other actions

Error Smoothing
- Can interpolate late actions
  - Create simulation for action
  - Avg into original simulation
- Continue until converge

Networking
The Perils of Error Correction
Physics: Challenge of Synchronization

• Deterministic bi-simulation is very hard
  • Physics engines have randomness (not Box2D)
  • Not all architectures treat floats the same

• Need to mix interpolation with snapshots
  • Like error correction in optimistic concern
  • Run simulation forward from snapshots
Physics: Challenge of Synchronization

- Deterministic bi-simulation is very hard
  - Physics engines have randomness (not Box2D)
  - Not all architectures treat floats the same
- Need to mix interpolation with snapshots
  - Like error correction in optimistic concern
  - Run simulation forward from snapshots

See today’s reading
Physics: Challenge of Authority

- Distributed authority is very difficult
  - Authority naturally maps to player actions
  - Physics is a set of interactions

- Who owns an uncontrolled physics object?
  - **Gaffer**: The client that set in motion
  - Collisions act as a form of “authority tag”
Summary

- **Consistency**: local state agrees with world state
  - Caused by latency; takes time for action to be sent
  - Requires complex solutions since must draw now!

- **Authority** is how we measure world state
  - Almost all games use a centralized authority
  - Distributed authority is beyond scope of this class

- **Synchronization** is how we ensure consistency
  - Pessimistic synchronization adds a sizeable input delay
  - Optimistic synchronization requires a lot of overhead