Lecture 12

Networking
<table>
<thead>
<tr>
<th>Consistency</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Do our games agree?</td>
<td>• What cheats are possible?</td>
</tr>
<tr>
<td>• Where do I see objects?</td>
<td>• View hidden data</td>
</tr>
<tr>
<td>• Where do you see them?</td>
<td>• Enter invalid states</td>
</tr>
<tr>
<td>• Who is authoritative?</td>
<td>• Improve player skill</td>
</tr>
<tr>
<td>• How to force agreement?</td>
<td>• How do we cheat proof?</td>
</tr>
<tr>
<td>• Do I wait for everyone?</td>
<td>• Technical solutions?</td>
</tr>
<tr>
<td>• Do I guess and fix errors?</td>
<td>• Community policing?</td>
</tr>
</tbody>
</table>
CS 3152: Game Networking Issues

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?

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
  - Need to understand basics before solving this

- This makes the game less responsive
  - Need some way to compensate for this
### Networking Breaks into Two Phases

#### Matchmaking
- Service to find other players
  - Groups players in a session
  - But does not run session
- Why make your own?
  - Control user accounts
  - Implement skill ladders
- 3rd party services common
  - Apple GameCenter
  - GooglePlay API
  - Unity’s server classes

#### Game Session
- Service to run the core game
  - Synchronizes player state
  - Supports minor adds/drops
- Why make your own?
  - Must tailor to your game
  - You often have no choice
- Limited 3rd party services
  - Often just a networking API
  - For limited class of games
  - **Examples**: Unity, Unreal
## Networking Breaks into Two Phases

### Matchmaking
- Service to find other players
  - Groups players in a session
  - But does not run session
- Why make your own?
  - Control user accounts
  - Implement skill ladders
- 3rd party services common
  - Apple GameCenter
  - GooglePlay API
  - Unity’s server classes

---

### Game Session
- Service to run the core game
  - Synchronizes player state
  - Supports minor adds/drops
- Why make your own?
  - Must tailor to your game
  - You often have no choice
  - Limited 3rd party services
    - Often just a networking API
    - For limited class of games
    - **Examples**: Unity, Unreal

---

**Simplify if possible**

**Our main focus**
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)

- **AdHoc Servers**: The cheap but ugly solution
  - One app declares itself to be a server
  - Other apps type in the IP address of that app

- **Benefit**: cross-platform matchmaking
  - Only way for iOS to play with Android
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)

- **AdHoc Servers**: The cheap but ugly solution
  - One app declares itself to be a server
  - Other apps type in the IP address of that app

- **Benefit**: cross-platform matchmaking
  - Only way for iOS to play with Android

This is allowed, but not if you are going to release on store
Game Session: Part of Core Loop

Client
- Update
- Draw

Authority
- Update

Networking
Decoupling the Network Loop
Decoupling the Network Loop

Client
- Local Update
- Network Update
- Draw

Authority
- Update

Smooth local animation
- Possibly slower tick rate (10 fps)
- Should match the client rate
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
  - 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!
**Synchronization Algorithms**

- 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>• 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>• Player controls only avatar</td>
</tr>
<tr>
<td>• Simulation games</td>
<td>• All else approximated</td>
</tr>
</tbody>
</table>

- Best on fast networks
- Local LAN play
- Bluetooth proximity
- Or games with limited input
- Real time strategy
- Simulation games
## Synchronization Algorithms

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

**Networking**
- True State
Optimistic: Opponent State

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

Previous Network State
- State updates
- Current Network State

Server Update

Draw

Local Update

Network Update

Networking

True State

Opponent actions
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
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