LECTURE 10
Announcements
2 down, 3 to go!

• As of today:
  – Engines are done
  – Games are “playable”
    • Think pre-minimum viable product

• Playtesting today: people should “get” what your game is about
No Credit Assignments Reminder

• No credit assignments must be turned in and graded complete by week after final 5
  – May 16th

• Don’t wait until that week!
  – What if you don’t get it first try?
  – We don’t want to give anyone an NC 😞
Playtesting moving forward

• You *must* have a /contrib project
  – Update it just before playtesting
  – Or whenever else you like
• 5 playtest signatures required each week
• Please let us know what your demo script is!
  – Should be contrib-link’ed, which will add it to everyone’s path!
Final3 rubric meetings!

• Same thing as final1 and final2 rubric meetings

• Only 1 rubric for this week
  – Final4 will depend on state of final3

• Goal of final3 is an MVP with no polish
  – Basically, your core gameplay is done and the fun/compelling parts are in place
QUESTIONS?
LECTURE 10
Networking
Networking

NETWORKING STRATEGIES
The Illusion

• All players are playing in real-time on the same machine
• But of course 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

• Things working against us:
  – 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
- ...but usually it’s bad
  - RTS: there are a million units
  - FPS: there are a million players
  - Fighter: timing is crucial
Modeling the World

• If we’re sending everything, we’re modeling the world as a uniform chunk
  – But it really isn’t!
  – Composed of entities, only some of which need input from a player
• We need a better model to solve these problems
Send Commands

- Model the world as local and shared data
  - Share player information, powerups, etc
  - Don’t need to share static level data
- Each player sends the other all actions that alter shared game world
- “Deterministic P2P Lockstep”
- Problem: everything must evaluate the same
  - Or else there are desyncs
- Problem: have to wait for all the other players’ commands
  - So everyone is limited by laggiest player

```
Player 1
State 1
  processP1Inputs()
  sendReceiveInputs()
  processP2Inputs()
State 2
Player 2
State 1
  sendReceiveInputs()
  processP1Inputs()
  processP2Inputs()
State 2
```
Client-Server Model

- One player is the authoritative server
  - Now we don’t have to wait for slow players, just the server
- Other player is a “dumb terminal”
  - 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

• But the server just sent a state that was 100ms in the past!
• What if games have diverged since then?
  – For instance, both players think they’ve collected a single powerup
• We can’t just replace our game world or we’ll lose commands from the local player
  – Client has to roll back the world and integrate commands since the last known good state
What about the server?

• Without rollback:
  – In an FPS, would need to lead shots because the server won’t register shot until after delay

• With rollback:
  – The target could be shot after they think they’ve taken cover
  – Or we could delay the server player as well…

• Need to think carefully about both technical requirements and game impacts of any networking model
Masking the Timewarp

- Problem: laggy players experience this jump often
- Solution: if the server usually sends states from 100ms ago, run the client 100ms behind
- Turns a jumpy experience into a smooth, only slightly slow one
  - Very useful if relative timing of commands is important
Edge Cases

• What if...
  – The client disconnects
  – The server dies
  – The client goes insane and sends gibberish
  – The client loses internet for 30 seconds
  – The client is malicious
  – The client changes IP address

• Handling errors well is vital to player experience
Elegant Disconnects

• Handle 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!
Networking Strategies

QUESTIONS?
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 (websites, databases, etc)
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 (streaming, etc)
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 important data, UDP sends timely data
- Best choice varies by project
  - (for naïve version, TCP is fine)
C sockets

• So much more than we want to cover in class
• Pros:
  – Full control over network throughput
  – Worth 5+ points
  – You will learn a lot
• Cons:
  – Oh so much more complicated
  – Will require multithreading, synchronization, and an incredibly well thought out design

• Start with CS033’s snowcast project
C++ QSockets

• Qt has QTcpSocket and QUdpSocket!
• Pros:
  – Far easier to set up than standard sockets
  – Convenient blocking, non-blocking IO calls
• Cons:
  – Still sending/reading bytes
  – Still need multithreading, synchronization, and a good design
  – Ton of error checking required

• Better, but still not perfect. So…?
Networking - Implementation

QUESTIONS?
The RakNet library

- Open-source games networking library
  - Recently bought by Oculus!
  - Plugin-style
- Used by some *really* legit engines
  - Unity, Havok, Minecraft
- Find it here: http://www.jenkinssoftware.com/
The basics

• Basic client-server or P2P connections
  – Read and write threads made for you!

• BitStreams that can serialize:
  – Primitives (char, int, long, etc…)
  – !!! Structs !!!

• Basic packet objects with metadata
Isn’t that everything…?

• Now sending data is easy...great!
• Still have to...
  – Pick what to send
  – Pick when to send
  – Interpret what is sent
• What if I have 1000 entities in my world?
  – Entire world may be too much data...
  – Need some complex ID system
• Gee, it would be great if...
RakNet does that too!

- Introducing ReplicaManager3!
  - Networked entities inherit from Replica3
  - RakNet gives you callbacks for all serialization events
    - onConstruct
    - onDestruct
    - onSerialize (each tick)
- *A LOT* of setup required, but works amazingly well
- Probably better to extend it a bit for simpler callbacks
Using ReplicaManager3

- Have entities override some “NetworkedEntity” class that does most of the setup
  - Most of it is the same for every object
- Determine where entities are made and destroyed
  - Client or server side?
- Override serialization methods
  - Feed stuff in/out of a BitStream in the right order!
- Register created/destroyed entities with RakNet
- ???
- Profit!
Sounds great!

- Since you decide what’s serialized, you can avoid sending things other clients don’t care about…
  - But what about things that don’t change often?
- VariableDeltaSerializer!
  - Only sends data that hasn’t changed since last tick!
  - More work now, less network throughput
- Space-time tradeoff probably worth it…
I’m Sold!

- Lobby system
- “Fully connected mesh” host determination system
- Authentication protocols
- Team management

- !!! Voice chat !!!
- SQLite3 databases
- And so much more…
In conclusion...

- RakNet is a beast.
- Pros:
  - Handles the nitty-gritty threads/sockets details for you
  - RM3 really simplifies the design process
  - Lets you focus more on engine design
  - Fast. We got 16 clients at ~30 FPS
- Cons:
  - A lot of setup required for RM3
  - You won’t learn as much 😞
- Want 3-4 networking points but not as interested in low level stuff? Use RakNet
QUESTIONS?
LECTURE 13
C++ Tip of the Week
C++ Tip of the Week

TEMPLATES
Templates

• Templates are the C++ way of making a function or class generic
• The goal of templates is to avoid re-writing code for multiple variable types
  – Any math functions that should work on multiple types of numbers
  – Classes that act as specialized containers for other classes
Template Functions

template <class type> type add(type a, type b) {
    return a + b;
}

• This function declares that it will be using a template class ‘type’, which will be inferred by the compiler based on the arguments used

• The function relies on operator overloading — you can use it for any class that supports ‘+’
Template Functions

• Template functions are instantiated at compile time for every version necessary
  – Calling `add(1, 2)` will generate `add<int>(int a, int b);`
  – `add<double>(2, 3)` creates the version for `doubles`
  – `add<const int>(3, 4)` makes a `const int` version

• This will add to build time and executable size
Template Functions

template <class type1, class type2>
void print(type a, type b)
{
    cout << a << “, ” << b << endl;
}

• This function will produce versions for every combination of inputs given
• Explicitly calling print<double, double>(1.5, 2) will prevent the compiler from implicitly creating print<double, int> to handle it
Template Classes

• Usage is similar to functions:
  
  ```cpp
  template <class KEY, class VALUE>
  class MyMap {
  void insert(KEY key, VALUE value);
  ...
  }
  ```

• The compiler will generate the entire class definition for each templated type

• Instantiated class have no relation to each other — `MyList<float>` cannot work with `MyList<int>`
Template Classes

• Most functions within a template class will also be template functions

• The compiler will only generate the functions for a template type used by that type’s instantiation
  — If ‘MyList’ had a function ‘getListSum’, which relied on overloaded ‘+’, it would still compile for unsupported types if those MyList instances never call that function
Templates

• There are many more subtle rules and features to explore if you make a complex template
• Be warned – because of the instantiation process, debugging info isn’t always useful
• More info:
LECTURE 13
C++ Anti-tip? of the week
Fun with templates

• Template metaprogramming
  – The C++ type system is Turing complete (i.e. can be used for computation)
  – Discovered by accident during C++ standardization
  – Compile-time programming: programs generating programs
  – Abuses template specialization

• C++ templates are a functional language
  – Recursion instead of iteration
  – Immutable variables
    • Create a variable that holds a type via typedef
    • Create a variable that holds an int via enum
C++ tip of the week

• Simple example: compile-time factorial

// Recursive template for general case
template <int N> struct factorial {
    enum { value = N * factorial<N - 1>::value };
};

// Use template specialization for base case
template <> struct factorial<0> {
    enum { value = 1 };
};

int result = factorial<5>::value; // == 5*4*3*2*1 == 120
Another example: compile-time linked list

```cpp
// Compile-time list of integers
template <int A, typename B> struct node {
  enum { num = A };
  typedef B next;
};
struct end {};

// Compile-time sum function
template <typename L> struct sum {
  enum { value = L::num + sum<typename L::next>::value };
};
template <> struct sum<end> {
  enum { value = 0 };
};

typedef node<1, node<2, node<3, end>, end> >> list123;
int total = sum<list123>::value; // == 1 + 2 + 3 == 6
```
C++ tip of the week

• **Drawbacks**
  - Much longer compile times (computation via template instantiation is inefficient)
  - No debugger, only page-long error messages
  - Turing completeness brings the halting problem

```cpp
// This code will infinite-loop the compiler
template <typename T> struct loop {
    loop<T*> operator->();
};
loop<int> i, j = i->fail;
```
Final 2 Playtesting