An intro to:

Real-Time Multiplayer Game Development

By Lital Novosolov Natan

About Me

- Pirate Kings developer at Jelly Button Games
- Loves the network
- Loves games
- Loves the code

Jelly Button Games

LETTING PEOPLE

MINGLE

THROUGH GAMES

Located in
Tel-Aviv

Founded
2011

39
Jellys

First Title:

Pirate Kings

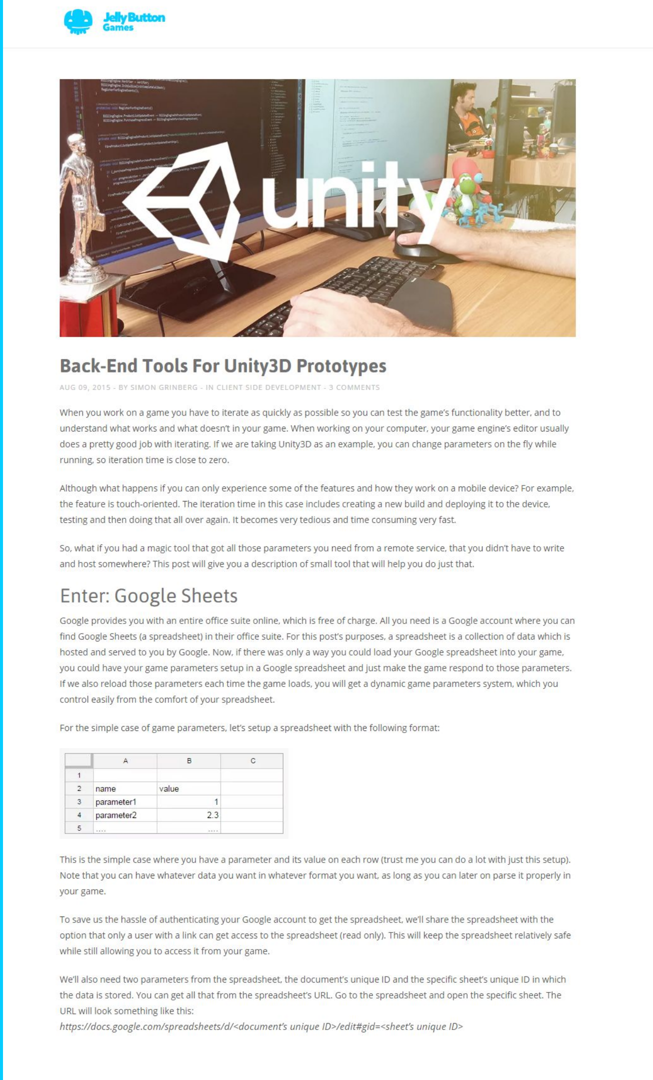
100% original, in-house production

50,000 new installs each day
90% viral/organic

Top 20 grossing in over 20 countries

50M installs overall

Check Out Our *Blog*



JellyButton Games

Back-End Tools For Unity3D Prototypes

AUG 09, 2015 - BY SIMON GRINBERG - IN CLIENT SIDE DEVELOPMENT - 3 COMMENTS

When you work on a game you have to iterate as quickly as possible so you can test the game's functionality better, and to understand what works and what doesn't in your game. When working on your computer, your game engine's editor usually does a pretty good job with iterating. If we are taking Unity3D as an example, you can change parameters on the fly while running, so iteration time is close to zero.

Although what happens if you can only experience some of the features and how they work on a mobile device? For example, the feature is touch-oriented. The iteration time in this case includes creating a new build and deploying it to the device, testing and then doing that all over again. It becomes very tedious and time consuming very fast.

So, what if you had a magic tool that got all those parameters you need from a remote service, that you didn't have to write and host somewhere? This post will give you a description of small tool that will help you do just that.

Enter: Google Sheets

Google provides you with an entire office suite online, which is free of charge. All you need is a Google account where you can find Google Sheets (a spreadsheet) in their office suite. For this post's purposes, a spreadsheet is a collection of data which is hosted and served to you by Google. Now, if there was only a way you could load your Google spreadsheet into your game, you could have your game parameters setup in a Google spreadsheet and just make the game respond to those parameters. If we also reload those parameters each time the game loads, you will get a dynamic game parameters system, which you control easily from the comfort of your spreadsheet.

For the simple case of game parameters, let's setup a spreadsheet with the following format:

	A	B	C
1			
2	name	value	
3	parameter1	1	
4	parameter2	2.3	
5	....	....	

This is the simple case where you have a parameter and its value on each row (trust me you can do a lot with just this setup). Note that you can have whatever data you want in whatever format you want, as long as you can later on parse it properly in your game.

To save us the hassle of authenticating your Google account to get the spreadsheet, we'll share the spreadsheet with the option that only a user with a link can get access to the spreadsheet (read only). This will keep the spreadsheet relatively safe while still allowing you to access it from your game.

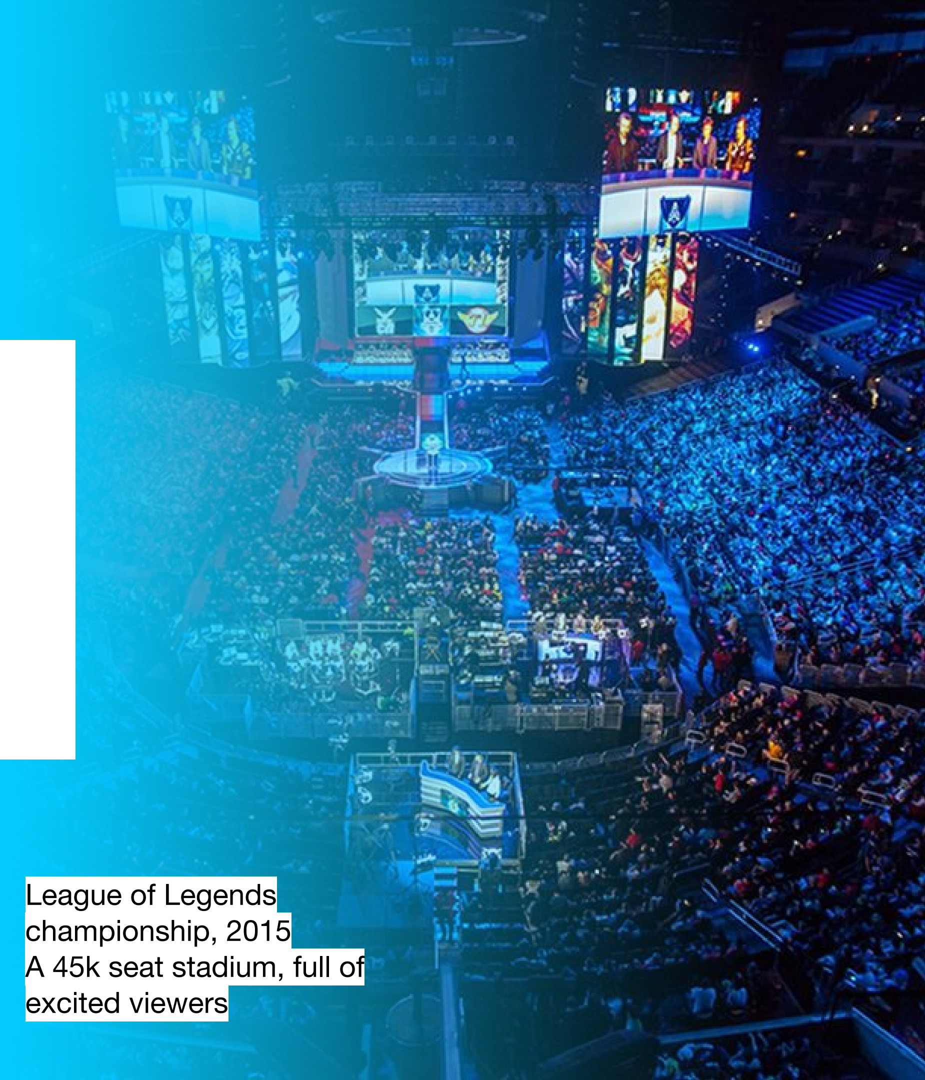
We'll also need two parameters from the spreadsheet, the document's unique ID and the specific sheet's unique ID in which the data is stored. You can get all that from the spreadsheet's URL. Go to the spreadsheet and open the specific sheet. The URL will look something like this:
https://docs.google.com/spreadsheets/d/<document's unique ID>/edit#gid=<sheet's unique ID>

We Will Talk About

- The world of multiplayer gaming and its challenges
- Different types of multiplayer gaming networks
- A few common techniques to deal with the lag away
- Sneak-peek at UNet, the new Unity networking framework

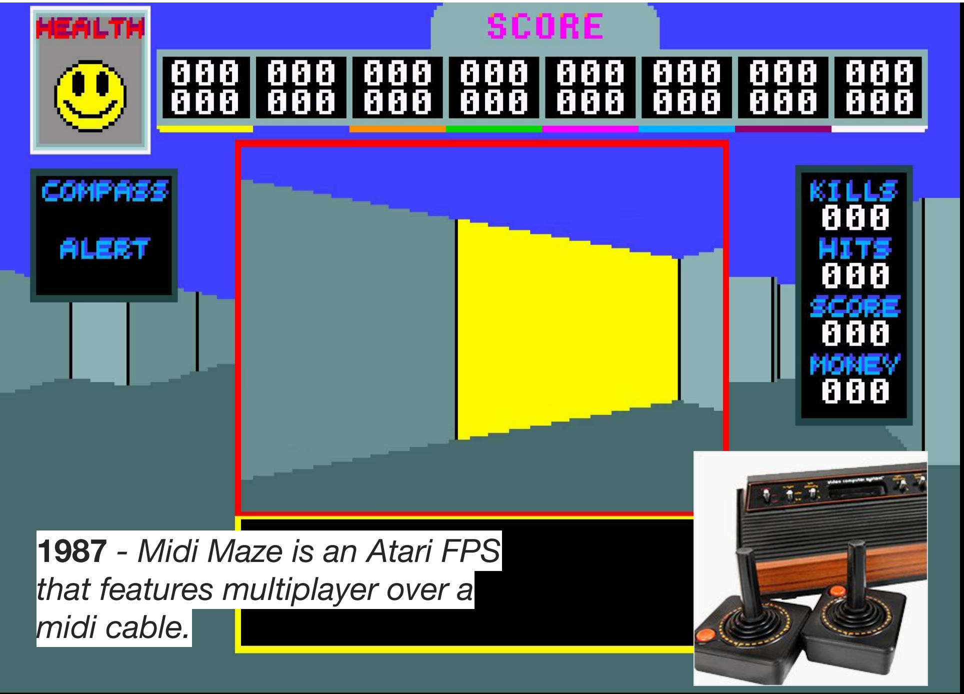
What Is So Special About Multiplayer Games?

Interaction With Others


League of Legends championship, 2015
A 45k seat stadium, full of excited viewers

Multiplayer Games

The beginning

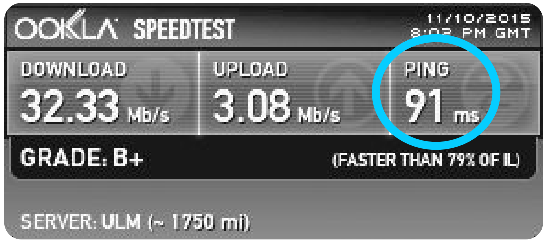


1987 - *Midi Maze is an Atari FPS that features multiplayer over a midi cable.*

1994 - Doom II released, featuring LAN and dial-up multiplayer

Multiplayer Games

Network Limitations



- Transfer of data takes time
- Data size is limited by bandwidth
- It's possible for data to get distorted, or never even arrive at all

Multiplayer Games

The internet era

1996 - “The earliest first-person shooter to use this [player prediction] technique was *Duke Nukem 3D*

-Wikipedia - Client-side prediction

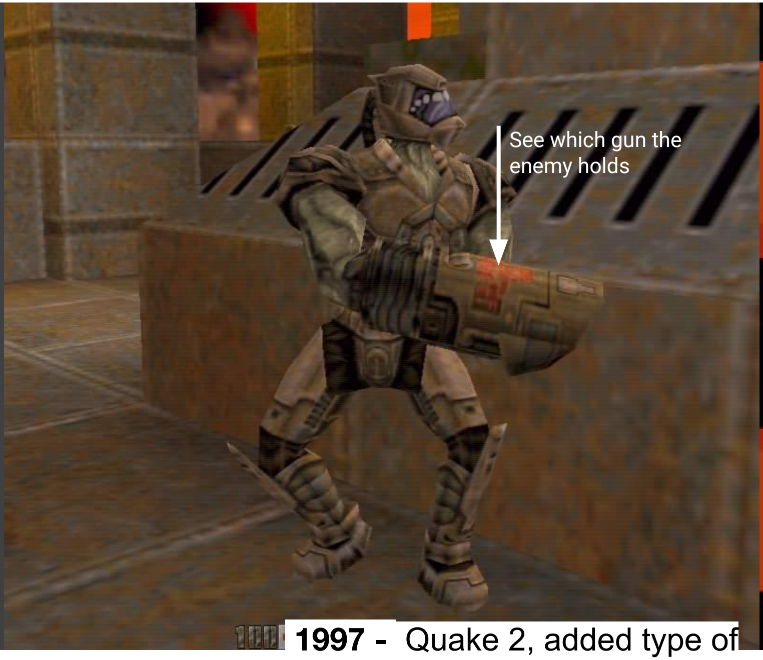
1996 - Id Software releases QuakeWorld, a network-enhanced version of the original Quake 1

Multiplayer Games

Enriching the multiplayer experience

Weapon is part of the player mesh

1996 - Quake 1,
Just model and animations



1997 - Quake 2, added type of weapon used by player

AW, my back!!
CADAVRE
CADAVRE

1999 - Quake 3 Arena, added vertical aiming angle

Types of Networks

In multiplayer games

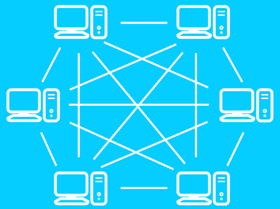
- Local multiplayer: local-machine, does not face any lag or bandwidth issues
- Turn based games: do not care about long delays
- Peer to peer: game clients communicate with each other in multiple channels
- Authoritative server: game clients communicate with a 3rd party that manages the game

Types of Networks

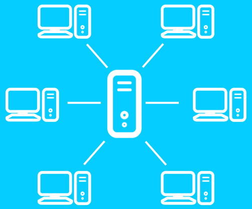
Peer-to-Peer VS Authoritative Server

Peer-to-Peer

- Direct (*faster*) communication
- Prone to desync
- Prone to cheating



Authoritative Server



- Connect through 3rd party (*slower*)
- One authority, less desync
- Less prone to cheating

Authoritative Server Is Your Game-Master

- All players connect to a 3rd party server
- The server acts as the “game master”
- Accepts input from the players
- Computes and dictates the state of objects and the world back to the players

Understanding State

- You are in the center of the room
- You are facing east, looking up
- You are holding a shotgun
- You are wounded
- You still have a bit of armor left

```
struct State
{
    Vector3 Position;
    Vector3 LookDirection;
    int WeaponId;
    int Health;
    int Armor;
}
```

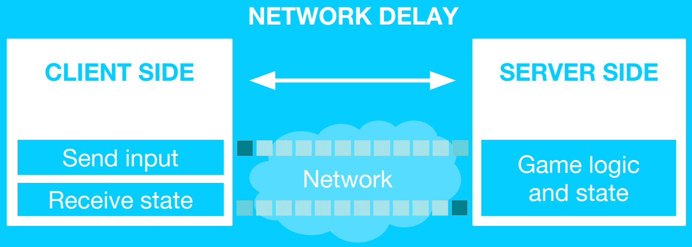
Understanding Input

- I would like to walk forward
- I am pulling the trigger, I want to fire
- I'm trying to duck for cover

```
struct Input
{
    Vector3 MoveDirection;
    Vector3 LookDirection;
    bool IsRunning;
    bool IsFiring;
    bool IsCrouching;
}
```

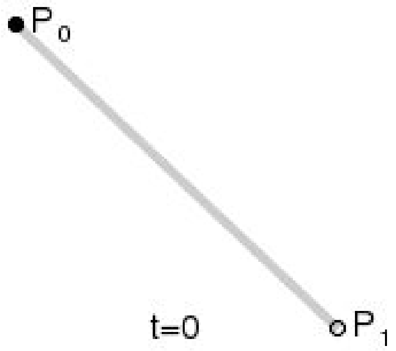
Authoritative Server Architecture

- Client sends input to the server
- Server receives input
- Server processes input
- Server sends state back to the client
- Client receives state and applies it



Processing object states on the client side

Handling Delays Between States



- There are delays between state updates that the client receives
- Use interpolation to overcome missing data between states sent from the server

Interpolation

(smoothing of motion)

Example client, naively snapping to the state

Example client, interpolating to new state

Don’t interpolate states too far apart, use *snapping*

Interpolation Won't Solve Everything

We still need to handle delays

- 0 MS: State (A) is sent from the server (You have 100% health)
- 40 MS: a rocket hits the player on the server, changes their state from (A) to (B)
- 80 MS: State (A) just arrived at the client, which is unaware of the rocket-hit

80 MS mark
FREEZE FRAME

Client:
I am perfectly fine, state (A)

VS

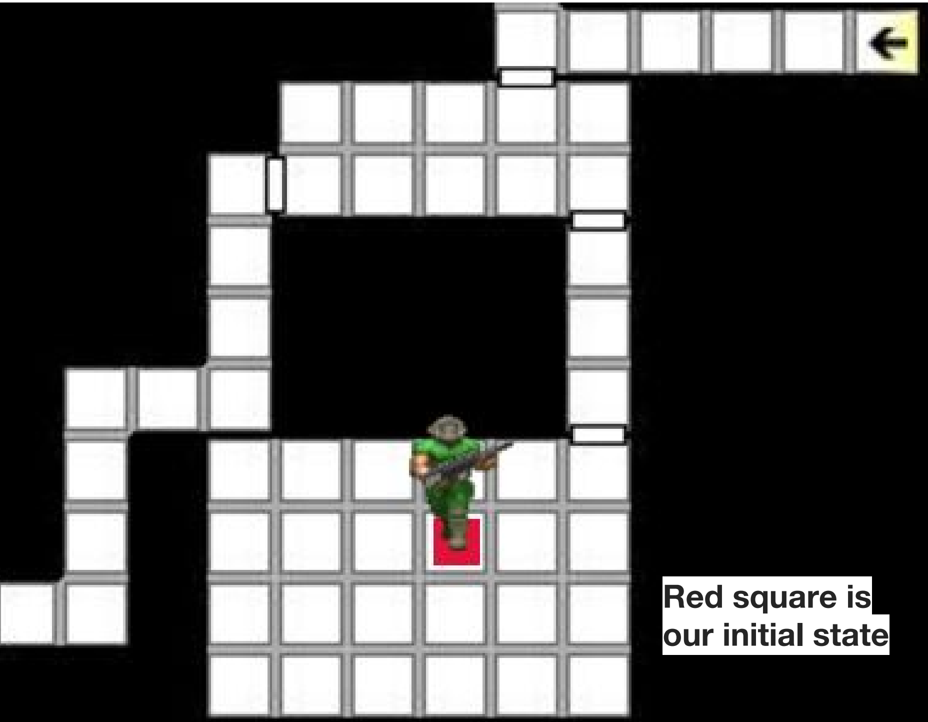
Server:
You are badly hurt, state (B)

The client gets state updates 80ms into the past and is still unaware of the damage the player took

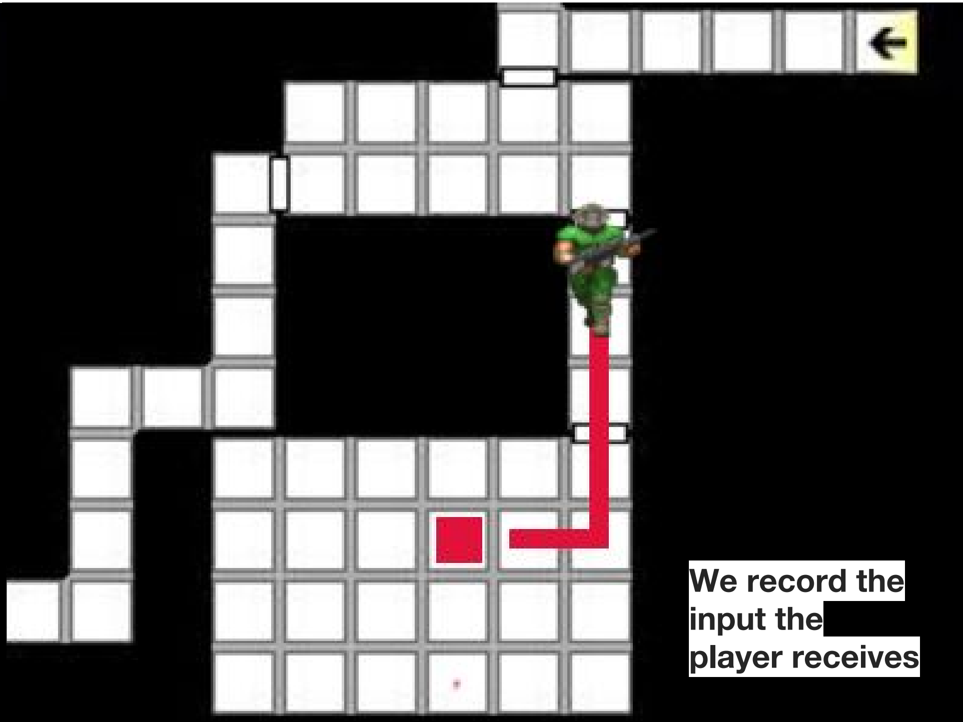
Rewind-Replay technique will address this

Rewind Replay

Understanding input-playback



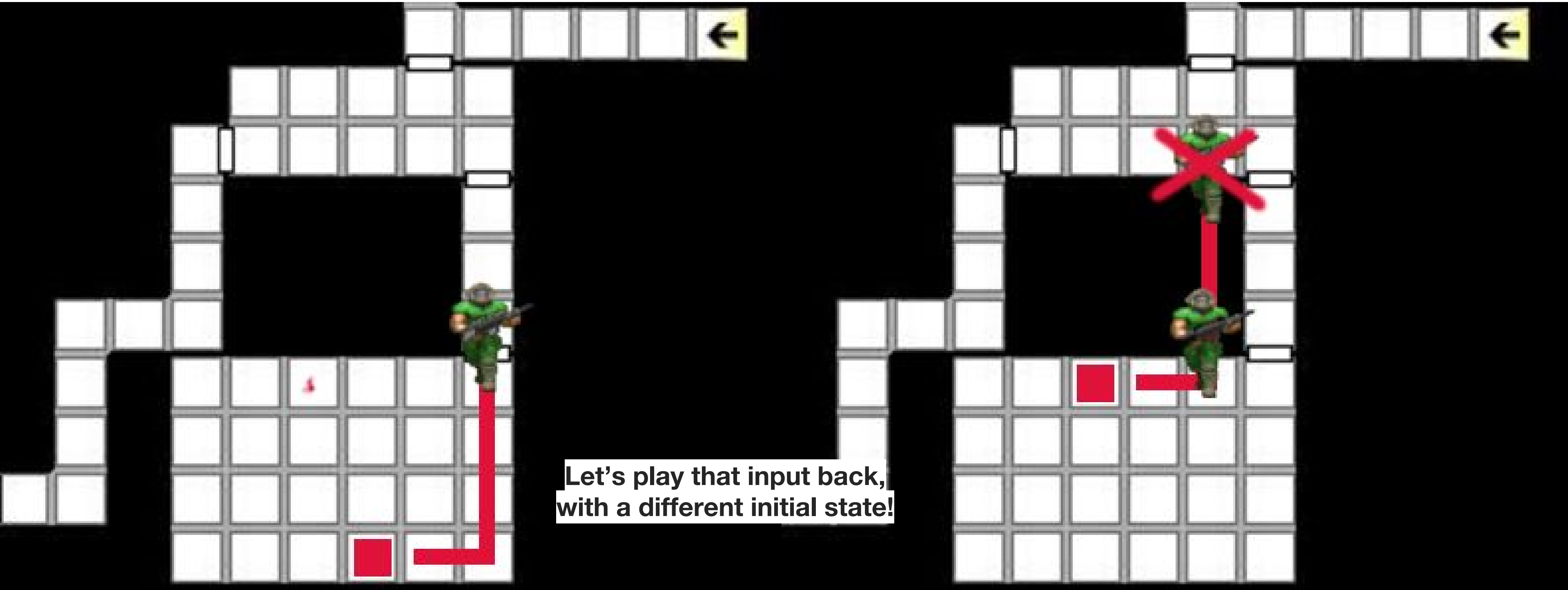
Red square is our initial state



We record the input the player receives

Rewind Replay

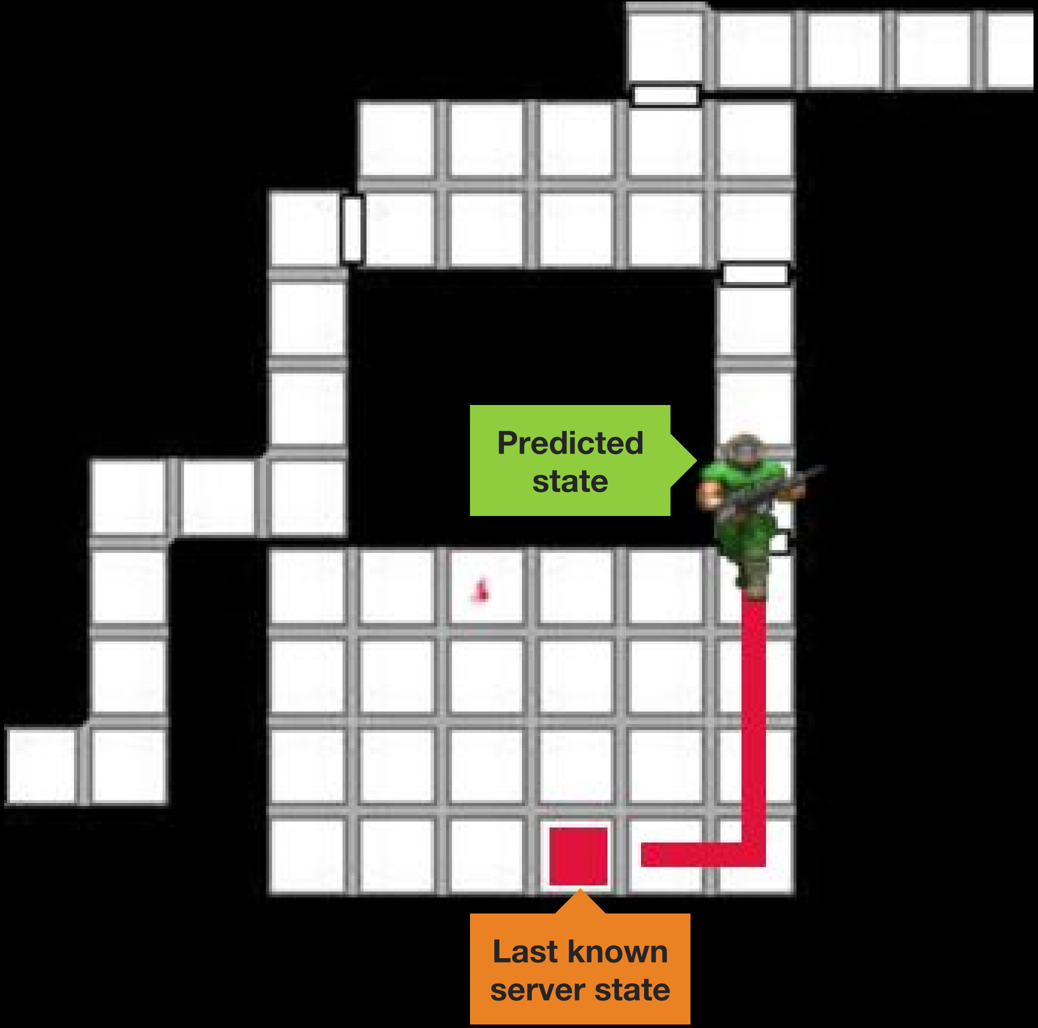
Understanding input-playback cont.



Rewind Replay

Understanding input-playback cont.

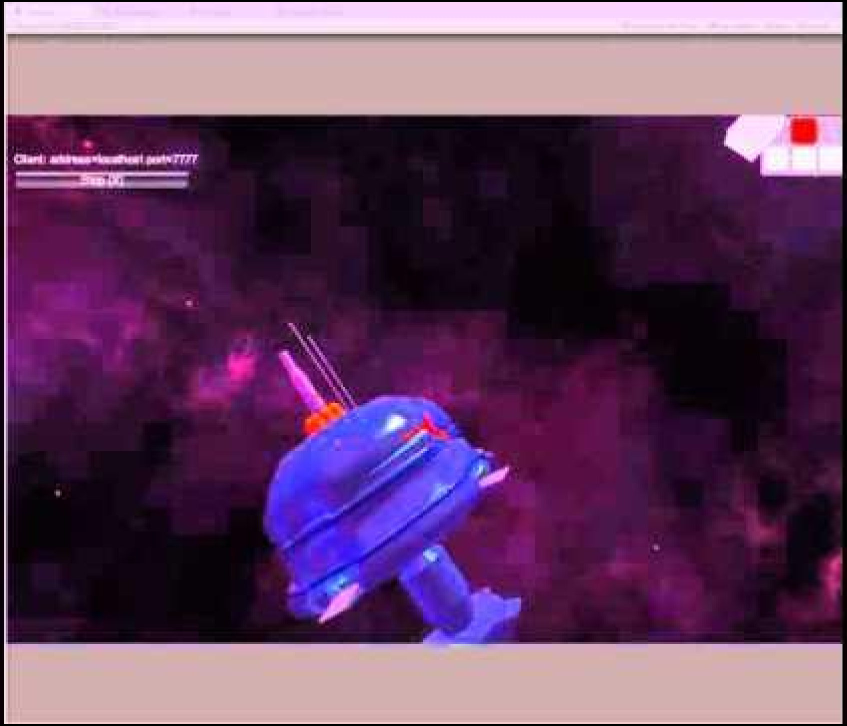
- The server sends us a state in the past
- We playback the input we recorded on top of this new “state-in-the-past”
- By doing this, we can “guess” our real-time state
- We don’t play all the input, just the part that is newer than the state we got!



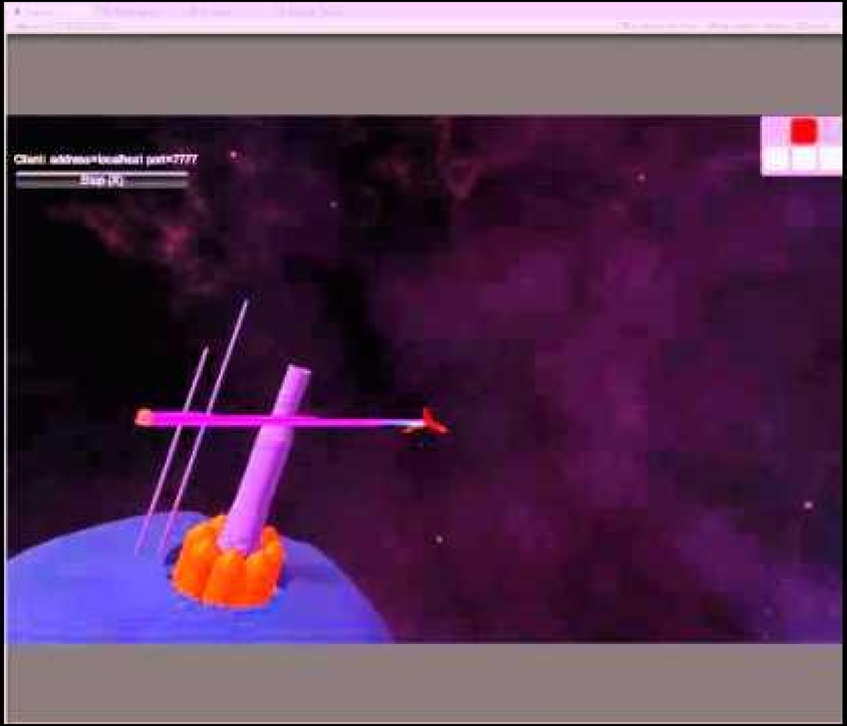
Rewind Replay

Comparing Rewind-Replay to simple interpolation

Test Conditions: 200ms LAG, 50% data loss



Interpolation without player prediction



Interpolation + Rewind-Replay

Handling Remote Objects

Remote Objects

Remote players:
Use other techniques

- Extrapolation (dead reckoning) - very inaccurate for non-predictable objects
- Use for objects you can pre-determine the behaviour for

Remote Objects

Remote players: **Use other techniques**

- Simply interpolate between states
- Extrapolation (dead reckoning) - very inaccurate for non-predictable objects
- Other server-side techniques

Remote Objects

Remote players:
Use other techniques

- There is no perfect solution
- Good elaboration in Wikipedia - “Lag”

Check Out The
Example Project!

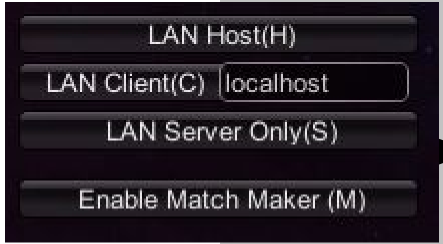
Unity 5 UNet

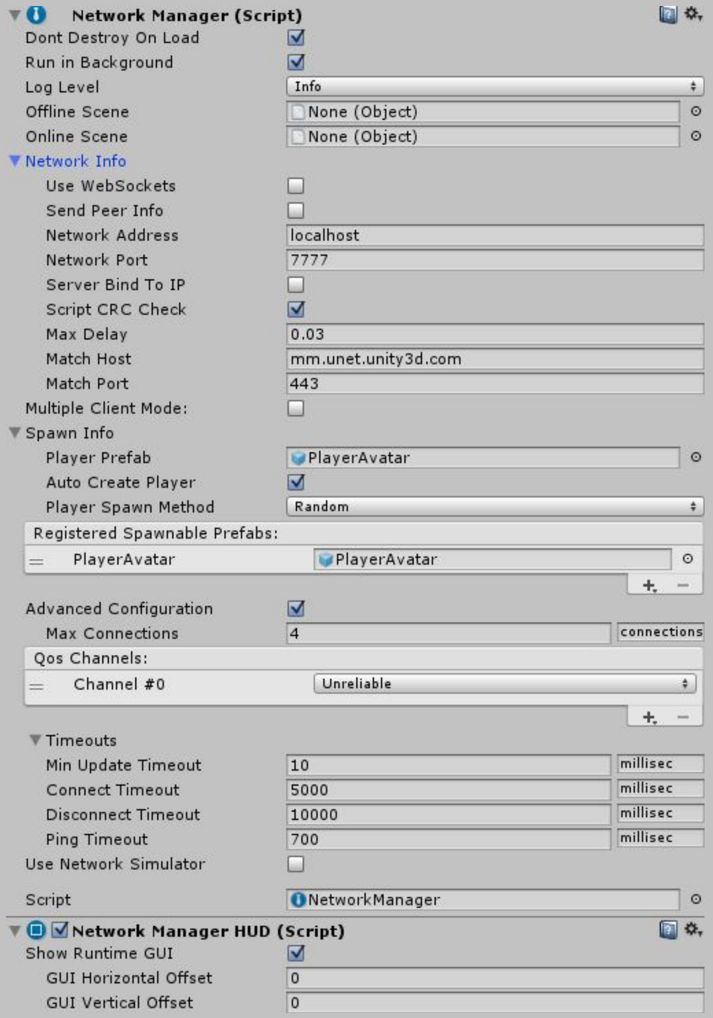
- A lot of improvement compared to past Unity versions
- Starting to look good at about ~5.2.2p2
- There's still a way to go
- Use same/shared codebase for server and client!

Unity 5 UNet

NetworkManager and NetworkManagerHUD

- Quickly set up a prototype
- NetworkManagerHUD will save you even from writing the small GUI to operate it
- Never use for actual releases.





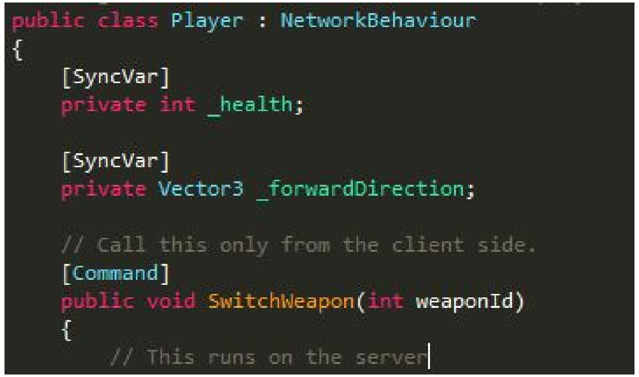
Network Manager (Script)

Dont Destroy On Load	✓	
Run in Background	✓	
Log Level	Info	
Offline Scene	None (Object)	
Online Scene	None (Object)	
Network Info		
Use WebSockets	☐	
Send Peer Info	☐	
Network Address	localhost	
Network Port	7777	
Server Bind To IP	☐	
Script CRC Check	✓	
Max Delay	0.03	
Match Host	mm.unet.unity3d.com	
Match Port	443	
Multiple Client Mode:	☐	
Spawn Info		
Player Prefab	PlayerAvatar	
Auto Create Player	✓	
Player Spawn Method	Random	
Registered Spawnable Prefabs:		
PlayerAvatar	PlayerAvatar	
Advanced Configuration	✓	
Max Connections	4	connections
Qos Channels:		
Channel #0	Unreliable	
Timeouts		
Min Update Timeout	10	millisec
Connect Timeout	5000	millisec
Disconnect Timeout	10000	millisec
Ping Timeout	700	millisec
Use Network Simulator	☐	
Script	NetworkManager	

Network Manager HUD (Script)

Show Runtime GUI	✓
GUI Horizontal Offset	0
GUI Vertical Offset	0

Network Behavior



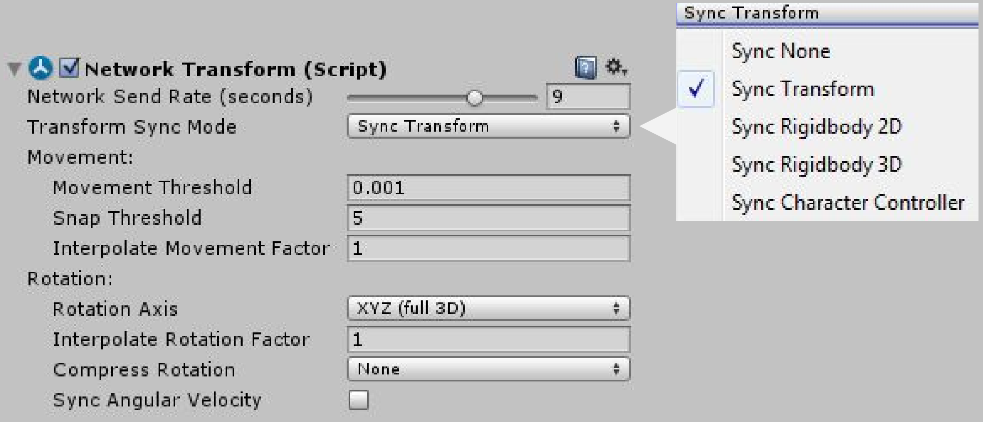
```
public class Player : NetworkBehaviour
{
    [SyncVar]
    private int _health;

    [SyncVar]
    private Vector3 _forwardDirection;

    // Call this only from the client side.
    [Command]
    public void SwitchWeapon(int weaponId)
    {
        // This runs on the server
```

- IsServer, IsClient, IsLocalPlayer
- [Command] / [RPC]
- [SyncVar] / [SyncEvent]
- [Client] / [Server]
- Spawn, OnStartLocalPlayer
- OnSerialize, OnDeserialize
- A lot of other goodies
  (read the documentation)

Unity 5 UNet

NetworkTransform

- Component that syncs transform states
- Can act on **Transform, RigidBody** (2d/3d) or **CharacterController**
- Includes basic **interpolation** that can be adjusted or turned off (and snapping threshold)
- Does not control children (there is **NetworkTransformChild** for that)

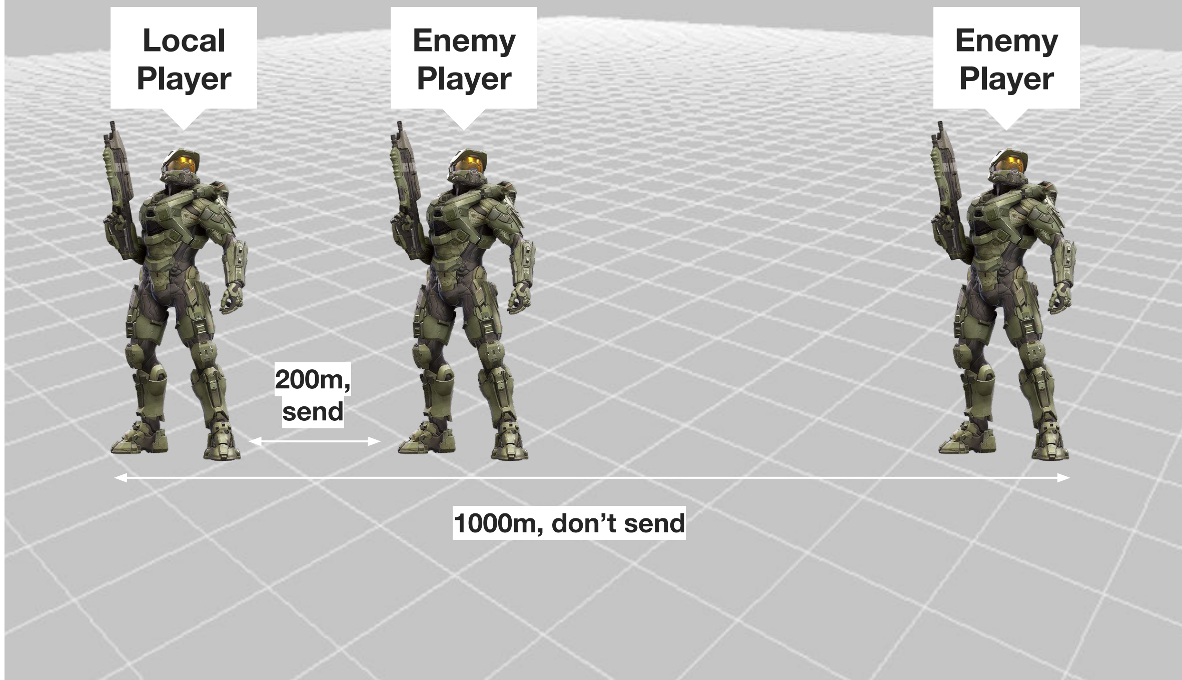

Network Transform (Script)

Network Send Rate (seconds)	9
Transform Sync Mode	Sync Transform
Movement:	
Movement Threshold	0.001
Snap Threshold	5
Interpolate Movement Factor	1
Rotation:	
Rotation Axis	XYZ (full 3D)
Interpolate Rotation Factor	1
Compress Rotation	None
Sync Angular Velocity	☐

Unity 5 UNet

NetworkProximityChecker

- Saves bandwidth by not syncing states of objects too far away
- A common practice in multiplayer games



Summary

- Real-Time multiplayer games are awesome
- If we want to provide a smooth real-time sensation in a multiplayer game, we should consider the limitations of networking when we develop our game
- There are many tricks to help simulate this sensation, we only went over a few here.
- UNet is a nice framework, care for bugs and stay tuned for patch releases
- Download our example project from github!

Jelly Button Games

THANK YOU!

Contact me for any further questions:

lital@jellybtn.com | *blog.jellybtn.com*

References

- Gabriel Gambetta - Fast-Paced Multiplayer
- Gaffer On Games - Game Networking
- Valve Developer Community - Source Multiplayer Networking
- Wikipedia - Lag
- Wikipedia - Multiplayer video games
- Wikipedia - Client-side prediction