Fax's Retrospectives and Prefabs

Sharing some of my experiences and assets with □YOU□

- Squat Racing Retrospective
- Prefabs
    - FaxAnalytics
    - FaxFallDamage
- UdonSharp Nitpicks

Squat Racing Retrospective

This is restrospective of my game for the VRCPrefabs valentine's jam 2021.

> *Note: This is somewhat incomplete and was written looong ago. Hope it inspires you nonetheless!*

The Good Stuff □□□□□

Overall, I'm quite happy with how Squat Racing turned out! If you've not tried it before, click here to visit the world.

> Squat Racing なるワールドの動画。
> スクワットしてる間カーブする乗り物にのってレースするゲームみたい。
> □pic.twitter.com/z5bctYrHhv
>
> — のーざん (@North369jp) February 15, 2021

> My entry into the @VRCPrefabs World Jam combines SQUATTING and RACING! @thenosjo helped with 3D modeling, so join us this weekend for a LOVELY WORKOUT ❤❤□□□twitter.com/aAYSRx4pLH
>
> — Fax (@Faxmashine) February 10, 2021

Devs Together Strong

When I first created squat racing, Nosjo created 3D models and occasionally took a look at my code - though we mostly worked separately.

Three months later I wanted to improve the project. And to my surprise, it was in a *beautiful* state. Comments everywhere. A nice scene hierarchy. No editor warnings. Having someone peek over my

shoulder seems to have affected the way I work in a very positive way - even though it was mostly a solo project!

Of course it's possible to work cleanly when working solo - but it can be tough to be disciplined during a game jam. My most recent world, Draw. Guess. Panic?!, shipped with many networking-related issues. And now, mere *weeks* later, they seem impossible to fix due to band-aid fixes and no documentation. Not only did that make the world worse, it was also a bit of a downer.

In 'Draw. Guess. PANIC?!' You better not disconnect, or things might break..!

TL;DR: Collaborating with someone made me work more cleanly.

Simple, Strong Core Idea

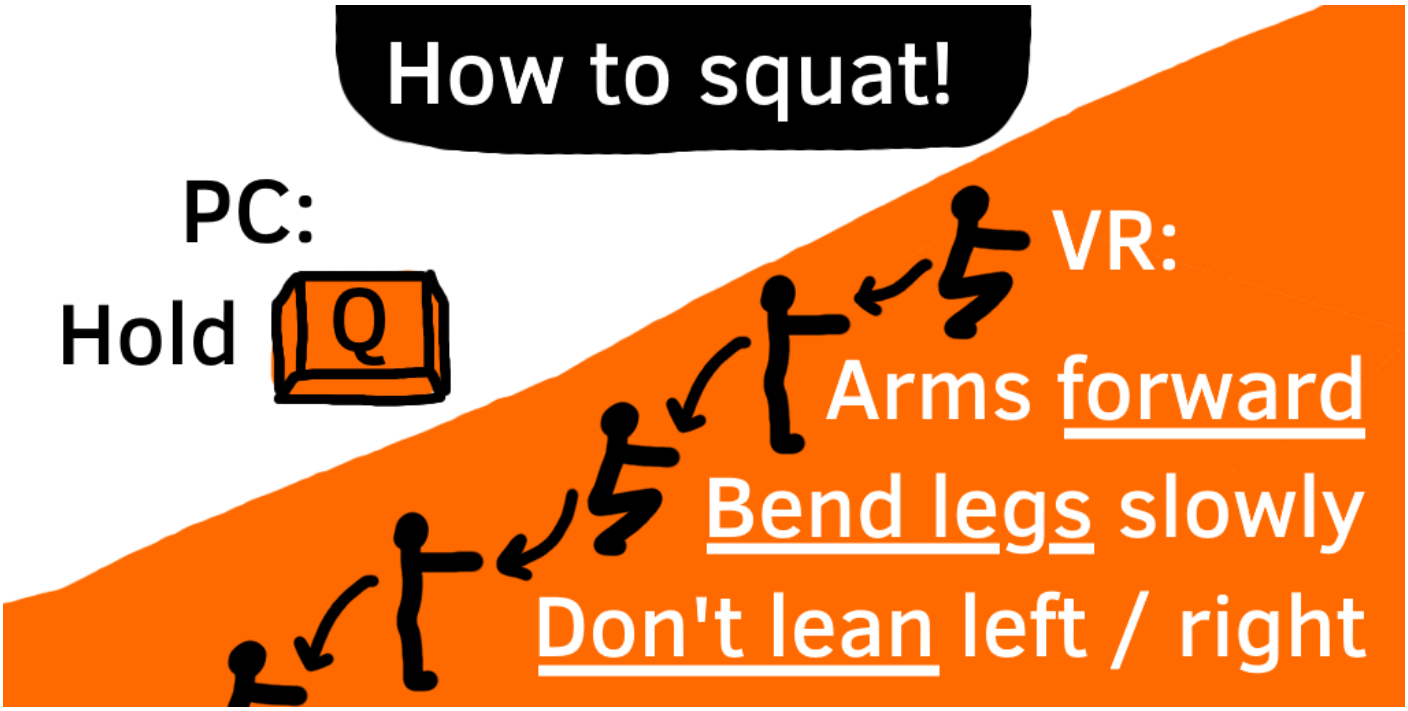
How do you do an exercise *game* in VR? Some users like to watch a video and follow along, but I wanted to create something more interactive.

The simplest thing I could think of was tracking the player's head position and forcing them to squat. Petey was quick to let me know that Squat Gym does exactly that. With Squat Racing I tried to utilize that same mechanic and to turn it into a game! With twists, and turns, and all that.

Squatting is easily explained, and everyone know what a race is. "Squat Racing" seems like a magic phrase that turns a lot of heads. Most players seem to have a good time! Having a low skill floor and high skill ceiling was great for keeping players motivated.

Creating Squat Racing was relatively simple, so I was able to iterate on my prototypes. Which was perfect for a game jam!

TL;DR: The concept was simple, engaging and easy to implement.



The Bad Stuff □□

Clear Goals

Most players who play squat racing do a single lap, say they enjoyed it, and leave the world. They don't try to get a good time, try to do multiple laps or discover the secret bonus-part of the race track.

The stats: Only around 30% of players who join the world complete a lap. Around 10% of them attain a solid grasp of the game's mechanics. That's... less than I hoped for. And it became obvious quickly into the project's development!

Now, on the bright side players say they have a good time! Especially those who take the time to understand how Squat Racing works. But I seem to have underestimated how difficult it is to learn the game!

Now, the core rules can be expressed in four sentences:

1. Press both triggers to start racing
2. Stand up straight to go straight
3. Crouch to turn
4. The direction you turn flips after every crouch (left - right - left - right)

Most players would then proceed to press both triggers, endlessly bump into walls or keep spinning in a circle. The panic sets in, and they forget the instructions (even though they're right in front of them!)

What Squat Racing is missing is a **safe environment to learn the rules**. Players start in a walled-off area, which is covered in grass to slow them down. But that still meant that players had to do too many tasks at once. Find the tutorial, read it, try it out, try not accidently move away - ideally I would have placed players in an endless void to let them try out the mechanics safely.

Tangled cables

VR players don't all have wireless headsets. This can cause issues when you have a circular race track. Doing three laps can take as little as 90 seconds, so playing for 30 minutes twists your cable **20 times**.

Here's two things I did to mitigate this:

- Put up an in-game poster warning players to not get tangled (seen above)
- Turn the track into an 8-loop (added post-launch)

Neither is an ideal solution. And I learned from my mistakes! In Super VR Ball many levels are arranged to prevent the player from getting tangled. Still - unless wireless headsets become the norm, keep tethered users in mind!

In conclusion

I'd say it went quite well for a two week projects! But it's certainly not without its flaws, and I'm sure there's untapped potential here. I don't see a lot of fitness games on VRChat, but following my release of Squat Racing, VRChat created a special row for workout wolds! And a few weeks later, VRChat held a game jam for racing worlds.

If you'd like to listen to the soundtrack, visit my SoundCloud! And if you like, please leave a comment below. Be as candid as you can, I'm sure there's something to be learned from any feedback you might have.

Prefabs

FaxAnalytics

This prefab lets you track information about your world! For example:

- How many players fall off your world?
- Does anyone use my mirror?
- Where do they spend their time?
- Does anyone press that button you added?

Download

https://drive.google.com/file/d/1gil_RAtmSSXFbKeTxd6OovrGam3Pn1e3/

What's included?

> This prefab utilizes video players in an undocumented manner. Do not use video player maliciously, respect the TOS!

- A prefab for quickly adding analytics to your world
- An example scene
- A full guide on how to use the prefab inside Unity
- UdonSharp scripts for you to tinker with

Requirements

- UdonSharp
- A Google Form to send the data to (after installation)
- A video player (included in the prefab)

How does it work?

After installing the prefab, check the top of your Unity window. You should see a new tab named "FaxAnalytics". This tab contains the full guide on how to use the prefab.

In a nutshell:

1. Create a Google Form.
2. Create a pre-filled URL for your form.
3. Modify the URL to auto-submit as soon as it's opened. (The prefab editor window has a tool to automaticall do this for you!)
4. Insert the URL into a video player.
5. Keep the video player disabled until you'd like the URL to be opened. (The prefab has some tools for this, too).

... That's it! If you'd like to add your own analytics without my prefab, all you really need is a video player.

Questions?

Contact me via Discord, at Fax#6041 or faxmashine.com.

Prefabs

FaxFallDamage

When players fall from a great height, they ragdoll and make a sound.

> **Jank alert!** This prefab uses the very obscure and obsolete VRChat **combat system**. Use at your own risk! It probably won't work with more than 20 players.

Download

https://drive.google.com/file/d/16sCyf2Ffc62x-7QZSgCvf_1WhOzdiGVe/

Requirements

- UdonSharp
- Lack of sanity
- <20 player world cap

How to use

Drag the prefab into your world, configure the settings, done. You can configure the minimum fall height, and the sound that plays when a player falls. Post your favorite death sound in the comments!! Mine's "oof" □□□□

This prefab was made at 3 AM

UdonSharp Nitpicks

> I hope you enjoy reading this - but keep in mind that this isn't quite finished ✌

Here's some things which I consider good practice when writing U# code, in no particular order. These suggestions are *somewhat* personal, so please let me know what you think in the comments below the article! ❤

`UdonSyncMode`

By default, the sync mode of an UdonBehaviour is chosen in the Unity inspector. Which is fine, until you accidently set it to the wrong sync mode. Yikes!

Instead, use U#'s `UdonSyncMode` attribute to make sure your `UdonBehaviour` always uses the sync mode you want.

```
using UdonSharp;
using UnityEngine;

[UdonBehaviourSyncMode(BehaviourSyncMode.Manual)]
public class Foo : UdonSharpBehaviour
{
  // ...
}
```

> This will prevent you from changing the sync mode on different GameObjects reusing the same U# script.
> But... does anyone actually do that? 🤔

`[SerializeField]`

If you make one of your variables `public`, it'll be accessible from the Unity inspector. It'll *also* make it accessible from any other `UdonBehaviour`. Most variables *aren't* accessed outside their own class, so making variables `public` makes your classes needlessly messy.

The solution? Keep those variables `private` instead, and just add `[SerializeField]` like so:

```
[SerializeField] private string foo;
```

> You can also do the opposite: To hide a public variable from the inspector, use [HideInInspector].

`[Tooltip]` and `[Header]`

U# scripts can have long and messy inspectors. Why not make them look nicer with the `[Header]` and `[Tooltip]` attributes?

```
[Header("References")]
[Tooltip("Reference to the mirror GameObject.")]
[SerializeField] private GameObject mirror;
[Tooltip("Transform to move the mirror to.")]
[SerializeField] private Transform targetTransform;
```

Cache your `LocalPlayer`

If you're doing something with `Networking.LocalPlayer` frequently, consider caching the reference in start:

```
private VRCPlayerApi localPlayer;

private void Start()
{
  localPlayer = Networking.LocalPlayer;
}
```

... and re-using that reference instead of calling `Networking.LocalPlayer` directly.

```
private void Update()
{
  Debug.Log(localPlayer.GetPosition();
}
```

`OnDeserialization()` and `[FieldChangeCallback]`

Call `OnDeserialization` as owner

When the owner of a script uses `RequestSerialization()`, all players *except* the owner will execute `OnDeserialization()`. But if you override `OnDeserialization()` in your script, you can have the owner call it manually. This is useful if you'd like the owner of the script to execute `OnDeserialization()`, to replicate the same behaviour as remote clients.

For example, here is a simple script which the master can use to increase a `score` variable. The `scoreText` is updated for everyone in `OnDeserialization`.

```
[UdonSynced] private int score;

[SerializeField] private TextMeshProUGUI scoreText;

// The script owner can call IncreaseScore anywhere to increase the score by 1.
public void IncreaseScore()
{
  score++;
  RequestSerialization();
  OnDeserialization()
}

public override void OnDeserialization()
{
  scoreText.text = score.ToString();
}
```

`[FieldChangeCallback]` instead of `OnDeserialization()`

The approach above works fine for simple scripts. However, if you script has multiple synced variables, your `OnDeserialization()` method may become quite large. Consider using `[FieldChangeCallback]` with a Property instead. It is similar to the Udon Graph's `OnVariableChanged` node. For example:

```
[UdonSynced, FieldChangeCallback(nameof(Score))] private int score;

private bool Score
{
  get => score;
  set
  {
    score = value;
    scoreText.text = score.ToString();
  }
}

[SerializeField] private TextMeshProUGUI scoreText;

public void IncreaseScore()
{
  Score++;
  RequestSerialization();
}
```

`[FieldChangeCallback]` execution order issues

> I've lost a lot of time identifying this issue. Please learn from my mistakes!

If you have multiple networked variables, watch out when using `[FieldChangeCallback]`. When a `[FieldChangeCallback]` happens, **other networked variables may not be updated yet.** In other words, if your networked variables depend on each other, you may want use `OnDeserialization()` instead.

Notably, the owner of the script will not suffer from this issue. If you're seeing no issues as an object's owner, but outdated variables on remote clients, you may be running into issues with `[FieldChangeCallback]`'s execution order!

What else?

There's more I'd like to write about... Let me know if any of these sound interesting!

- Avoid Update(), use delayed events instead.
- Did you know that FixedUpdate() depends on your monitor's refresh rate? Yeah.
- Use synced variables, avoid networked events, if you can.
- FieldChangeCallback is great.
- Name your classes and methods properly.