Reviving Dead Online Worlds

Lilia Roo

Anthrocon 2025

Sticker © 2021 Recurrent

About Lilia Roo

- Furry since 2018
  - Wrote transformation stories!
- Software engineer
  - Graduated 2023, but been employed in industry for 5+ years
  - Cut teeth via small FOSS contributions beforehand
- Maintainer of sewifurs.org since 2021
- Maintainer of Xfburn since 2022
- Developer for Weasyl since 2024

Fursuit by Luno Wroo
Photo by Mallory Gwynn

Problem statement

Online-only games, or beloved versions of them, will go defunct...

...but we can bring them back through *science*!

Outline

Using an example project:

- Decompilation
- Renaming symbols
  - Cross-reference code with fan documentation
- Reverse engineering the network protocol
- Writing our new server!

Before you begin!

Every project is different

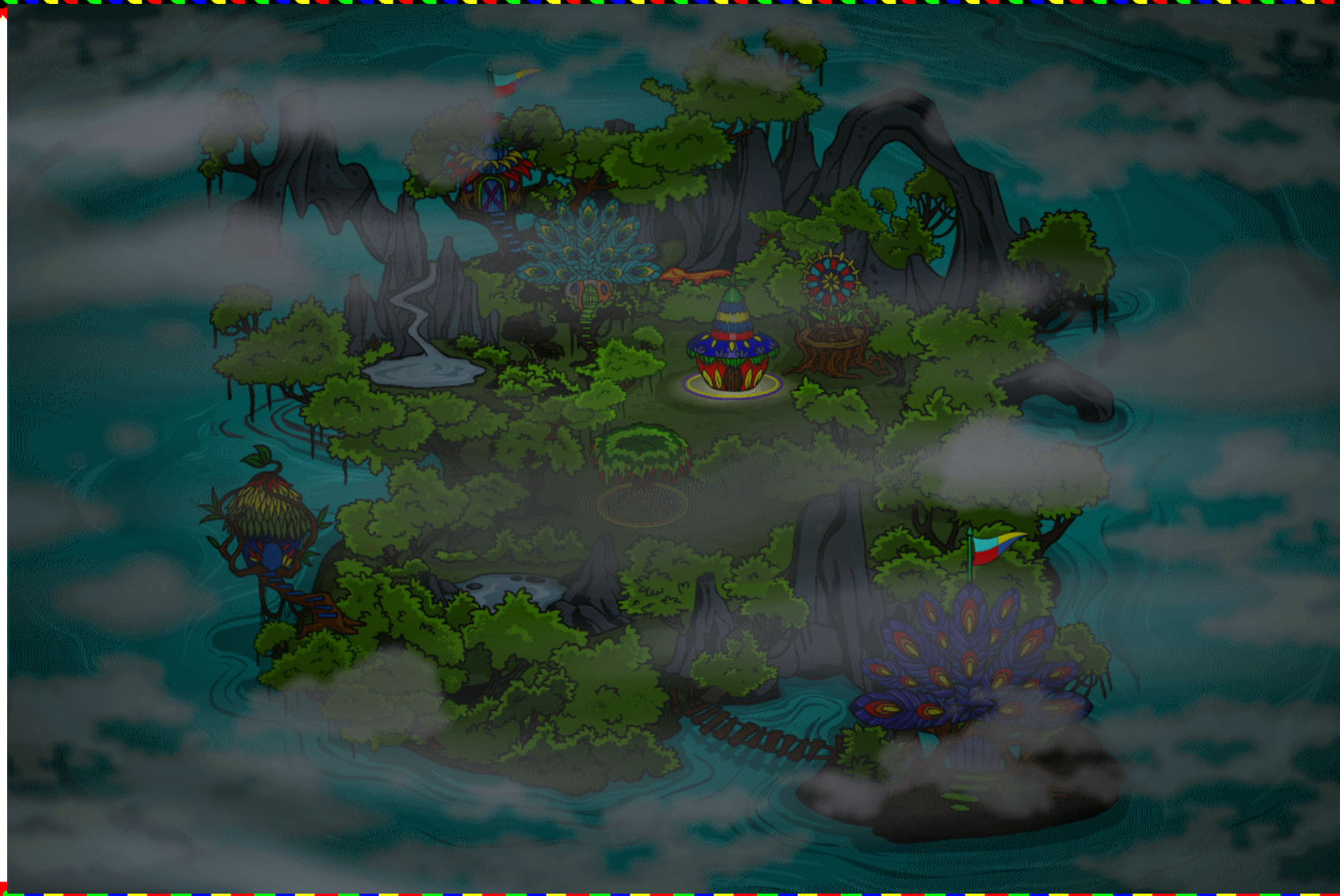
- Different programming languages
- Different libraries
- Some games are obfuscated
- Some games hate debuggers
- Some games have anti-cheat
- Does fan-written documentation exist?

Our example project

Neopets Mobile

- J2ME, 2006 – 2009
- Developer: In-Fusio
- Could sync progress with main website
- Only way to access Lutari Island!

Lutari Island



Decompilation!

- Used Vineflower (formerly Quiltflower)
- Bytecode → Source
- Symbols are obfuscated :(
  - This obfuscation uses normally-invalid duplicate names!

```
public final class p {
   public int a;
   private ByteArrayOutputStream a;
   private ByteArrayOutputStream b;
   private DataOutputStream a;
   private ByteArrayOutputStream c;
   private DataOutputStream b;

   public p(int var1) throws IOException {
      this.a = var1;
      this.a = new ByteArrayOutputStream();
      this.b = new ByteArrayOutputStream();
      this.a = new DataOutputStream(this.a);
      this.a(m.a + ";" + "2.8");
   }

   public final int a() {
      return 2 + this.a.size() + this.b.size();
   }
```

Decompilation!

- Ask decompiler to assign everything a unique name instead
  - The exact flag depends on your tools
- Easier to work with!

```
// $QF: renamed from: p
public final class class_2 {
   // $QF: renamed from: a int
   public int field_12;
   // $QF: renamed from: a java.io.ByteArrayOutputStream
   private ByteArrayOutputStream field_13;
   // $QF: renamed from: b java.io.ByteArrayOutputStream
   private ByteArrayOutputStream field_14;
   // $QF: renamed from: a java.io.DataOutputStream
   private DataOutputStream field_15;
   // $QF: renamed from: c java.io.ByteArrayOutputStream
   private ByteArrayOutputStream field_16;
   // $QF: renamed from: b java.io.DataOutputStream
   private DataOutputStream field_17;

   public class_2(int var1) throws IOException {
      this.field_12 = var1;
      this.field_13 = new ByteArrayOutputStream();
      this.field_14 = new ByteArrayOutputStream();
      this.field_15 = new DataOutputStream(this.field_13);
      this.method_37(class_17.field_154 + ";" + "2.8");
   }
```

Starting renaming

I gotta figure these out? Where do I even begin?

- Look for literal strings in functions
  - Mid-method log messages can help greatly
- Look for short, ‘leaf’ functions
  - Can more easily give a name based on what it does
- Look for recognizable constants
  - Check the game’s wiki; maybe it’s a value for a mechanic

Starting renaming

```
// $QF: renamed from: a () void
public final void method_79() {
   if (this.field_76 != 0) {
      this.field_77 = 0;
      this.field_75 = 0;
      class_8.method_67(this.field_84, 0);
      class_8.field_63 = 0;
      this.field_84.field_58 = 0;
      Thread var1 = new Thread(this);

      try {
         var1.start();
      } catch (Exception var3) {
      }
   }
}
```

```
// $QF: renamed from: a () void
public final void startLoadingImages() {
   if (this.loadingImageCount != 0) {
      this.loadedImageCount = 0;
      this.loadingImageId = 0;
      ImageLoader.setLoadedImages(this.parent1, 0);
      ImageLoader.loadErrorCount = 0;
      this.parent1.bytesLoaded = 0;
      Thread thread = new Thread(this);

      try {
         thread.start();
      } catch (Exception e) {
      }
   }
}
```

It’s like a puzzle, and you’re solving it piece-by-piece

Fan documentation helps!

Course Type	Level Range	# Codestones	Duration
Grasshoper	20 and below	1	2 hours
Basic	21-40	2	3 hours
Intermediate	41-80	3	4 hours
Adept	81-100	4	6 hours
Advanced	101-120	5	8 hours
Expert	121-150	6	12 hours
Master	151-200	7	18 hours
Grand Master	201-250	8	24 hours

Mystery Island Training School guide
From Jellyneo

```
if (activePetLevel <= 20) {
    trainingTimeLeft = 7200000L;
    trainingCost = 1;
} else if (activePetLevel <= 40) {
    trainingTimeLeft = 10800000L;
    trainingCost = 2;
} else if (activePetLevel <= 80) {
    trainingTimeLeft = 14400000L;
    trainingCost = 3;
} else if (activePetLevel <= 100) {
    trainingTimeLeft = 21600000L;
    trainingCost = 4;
} else if (activePetLevel <= 120) {
    trainingTimeLeft = 28800000L;
    trainingCost = 5;
} else if (activePetLevel <= 150) {
    trainingTimeLeft = 43200000L;
    trainingCost = 6;
} else if (activePetLevel <= 200) {
    trainingTimeLeft = 64800000L;
    trainingCost = 7;
```

Now we can give names to these!

Networking

- Search for URLs, IP addresses, etc.
  - Depending on the game, might be read from a config file, might be hardcoded
- Search for socket read/write
  - Depends on language, libraries, etc.
- Look at the call stack and referenced classes
- You found the networking code!

Networking

```
private static void callApi(int requestTypeId, String var1) throws Exception {
    StructuredWriter writer = null;
    Object var3 = null;
    ServerApi.initialize(11061, 61, gameName, language, "http://npprod-singtel.in-fusio.com/data-np/");
    if (requestTypeId >= 8) {
        ServerApi.setPhoneNumber("+12345556789");
    }

    switch(requestTypeId) {
        case 1:
```

This game has a hardcoded endpoint

What now?

- Need to figure out how the protocol works
  - Is it HTTP? Raw TCP? Raw UDP?
  - Is it synchronous request-response? Something else?
  - Is there a separate auth step?
  - What is the data format?
- Study the decompilation for a bit, taking notes
- To get a real sample, create a simple server that just reads from the socket and logs requests

Basic server quick notes

- If the game hits a specific domain, route that to 127.0.0.1 via your hosts file
  - Unix-ish systems: `/etc/hosts`
  - Windows: `%WINDIR%\System32\drivers\etc\hosts`
- Use your favorite programming language

Example network protocol

- HTTP, single endpoint
- Requests have plaintext headers, URI-encoded
- Actual body is in a custom binary format
- Each type of request or response has a 16-bit ‘tag’
  - For requests, this is the desired function
  - For responses, this is the meaning of the contents
- Has a buffer of numbers (byte/short/int), then strings
  - Items read in sequential order
  - A short in the number buffer can be an offset pointer in the string buffer
  - Type information not encoded

Structured data format

Following some plaintext headers...

Offset	Type	Meaning
0	short	Tag
2	int	Length-derived value: 8 * (2 + n + m)
6	short	number buffer len + 2 (n)
8	bytes	number buffer
n + 8	bytes	string buffer
m	short	0x00FF, end of structured data

Note that if tag == 0x8000 or 0x8064, a different format is followed:

Offset	Type	Meaning
0	short	Tag
2	int	(len(message) + 1) * 8
6	byte	0x00, start of message
7	string	error message to display in console

From my notes at: https://computers.huntertur.net/index.php/Neopets_Mobile

Example: Login request

```
00000000: 636c 6965 6e74 3d4e 454f 5045 5453 2f32  client=NEOPETS/2
00000010: 2e38 2667 616d 653d 3131 3036 3126 6365  .8&game=11061&ce
00000020: 6e74 6572 3d36 3126 6c61 6e67 7561 6765  nter=61&language
00000030: 3d65 6e0a a001 0000 01a0 0012 0000 0000  =en.............
00000040: 0040 0000 0040 0010 0018 0020 000e 4953  .@...@..... ..IS
00000050: 4f2d 3838 3539 2d31 3b32 2e38 0006 6164  O-8859-1;2.8..ad
00000060: 6164 6164 0006 676a 676a 676a 0000 00ff  adad..gjgjgj....
```

Login request:

Username: adadad

Password: gjgjgj

Example: Login request

```
00000000: 636c 6965 6e74 3d4e 454f 5045 5453 2f32  client=NEOPETS/2
00000010: 2e38 2667 616d 653d 3131 3036 3126 6365  .8&game=11061&ce
00000020: 6e74 6572 3d36 3126 6c61 6e67 7561 6765  nter=61&language
00000030: 3d65 6e0a a001 0000 01a0 0012 0000 0000  =en.............
00000040: 0040 0000 0040 0010 0018 0020 000e 4953  .@...@..... ..IS
00000050: 4f2d 3838 3539 2d31 3b32 2e38 0006 6164  O-8859-1;2.8..ad
00000060: 6164 6164 0006 676a 676a 676a 0000 00ff  adad..gjgjgj....
```

Headers: client = NEOPETS/2.8, game = 11061, center = 61, language = en
Tag: 0xa001 Length-based value: 416 (= 8 * (2 + 50))
Number buffer len + 2: 18 Number buffer: (16 bytes)
String buffer: (34 bytes) End of data: 0x00ff

Example: Login request

Number buffer:

```
0000  0000 0040  0000 0040  0010  0018  0020
```

(unused constants)

String buffer:

```
00 0e ISO-8859-1;2.8
00 06 adadad
00 06 gjgjgj
00 00
```

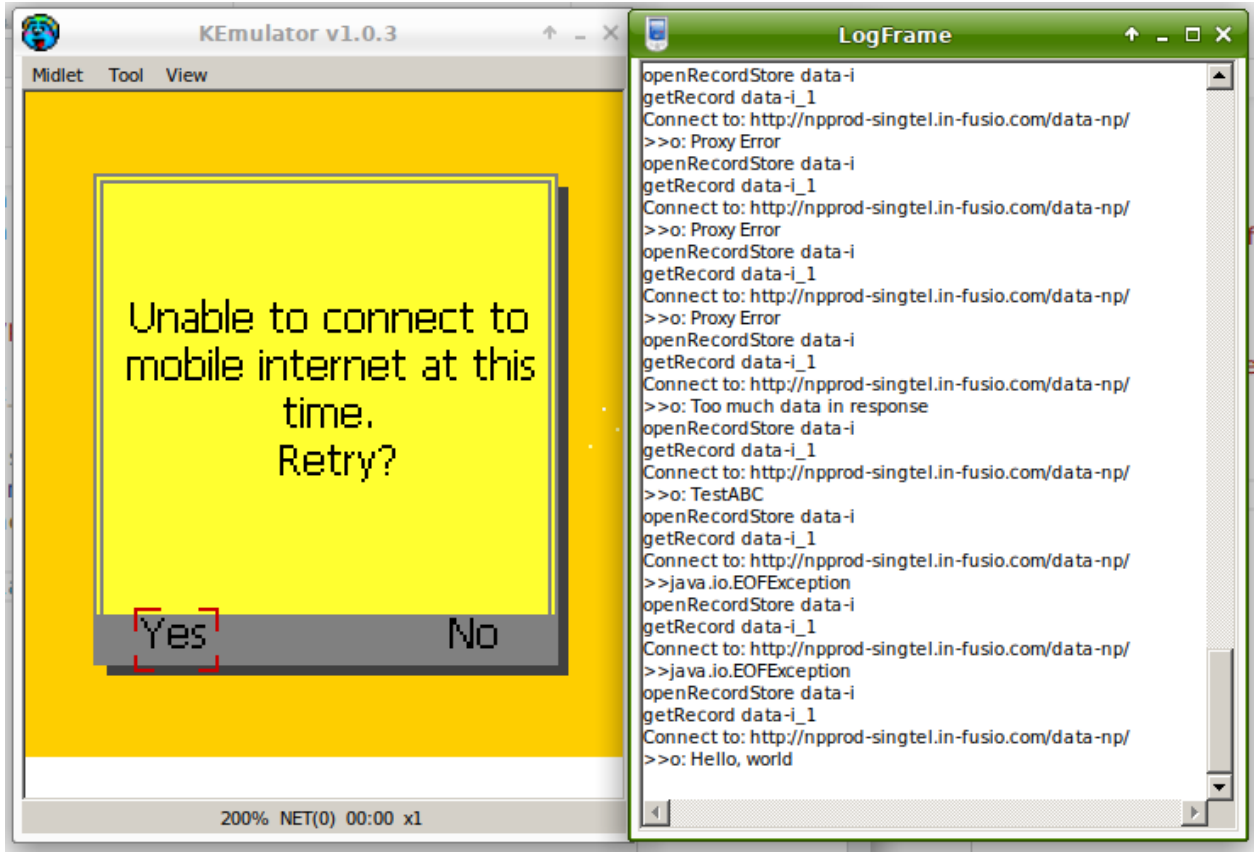
- Offset to encoding string
- Unused constant
- Unused constant
- Offset to username
- Offset to password
- Unused offset to blank string

Developers aren't perfect!
Don't stress about unused or constant values

Sending our first response

- Take a look over what you've discovered so far
- What's the easiest thing to implement first?
  - Always returning a game-recognized error on login?
- Goal: Make the client do something other than show a generic network error
  - This proves we figured out the basic interface of the API

Generic network error



Example: Error response

```
0aa0 0000 0000 4000 0800 0000 2900 0000 ff
```

Headers: None (newline-terminated)
Tag: 40960 (0xA000) (sync status)
Length-derived value: 64 (= 8 * (2 + 6 + 0))
Number buffer len + 2: 8
Number buffer: 41 (int), pointer to blank string
    (41 → user already logged in)
String buffer: just a blank string, so nothing
End of data: 0x00FF

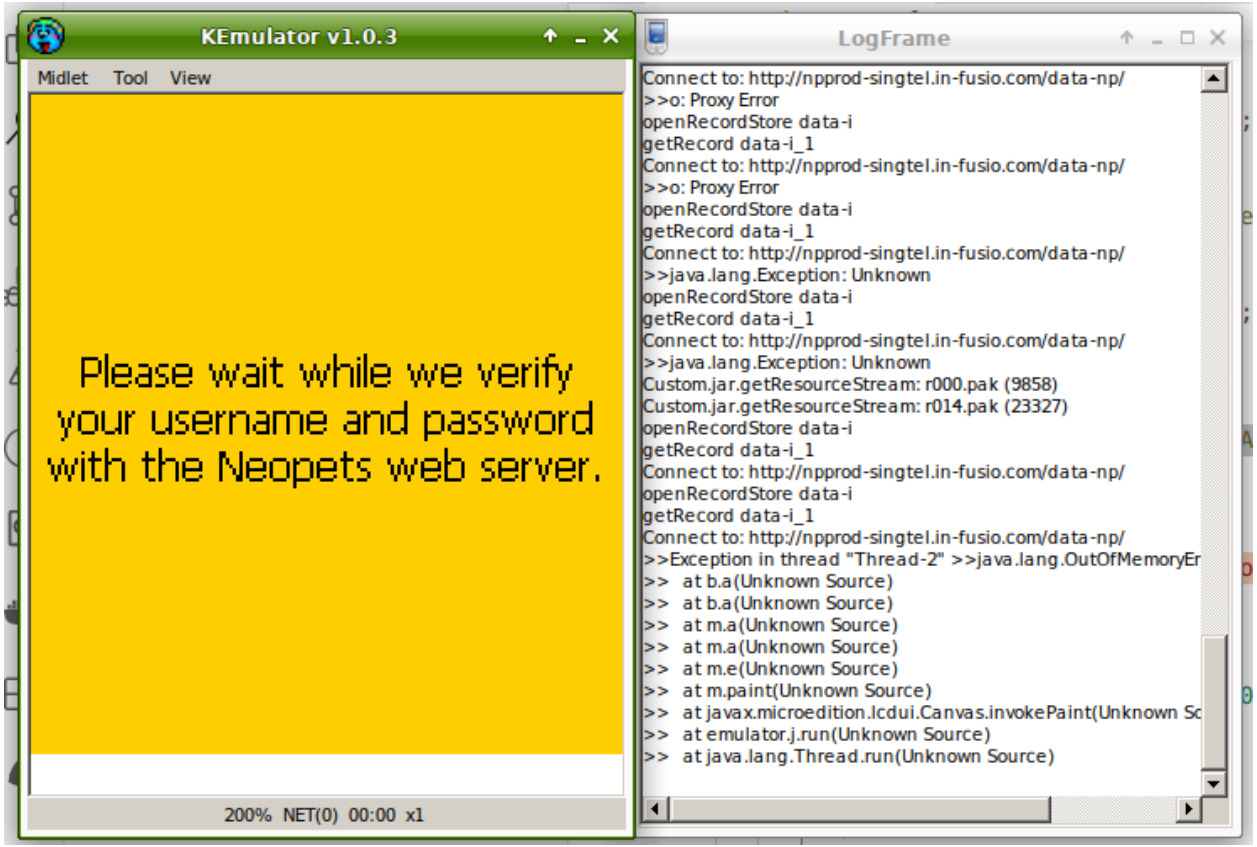
Game recognized error

- Even if we semantically responded with an error, the response itself is valid.
- Therefore, it’s working!

Building out the server

- It’s alive!
- Keep analyzing code and renaming symbols to understand other API calls’ interfaces
  - 80/20 rule: you’ll likely get 80% of the client functional by implementing 20% of the API
- Don’t worry about data persistence while prototyping

Client debugging can suck



What next!

- Data persistence
  - Let people truly create accounts
  - Store their game state on disk

Don't let your work go to waste!

Even if you don't finish, people would love to see what you uncovered!

- Any data specs, even partial, help immensely
- Future people could reference your notes!
- Found unused stuff? Share with tcrf.net
- Consider open-sourcing your new server

Thanks for coming!

Have any questions after the end of the hour? Contact me here!

- fursona.directory/@LiliaRoo
- FA: hukaulaba
- Weasyl: liliaroo
- Bluesky: @liliaroo.furwaukee.org
- Mastodon: @hukaulaba@transfur.social
- Discord, Telegram: liliaroo
- Email: hukaulaba@gmail.com