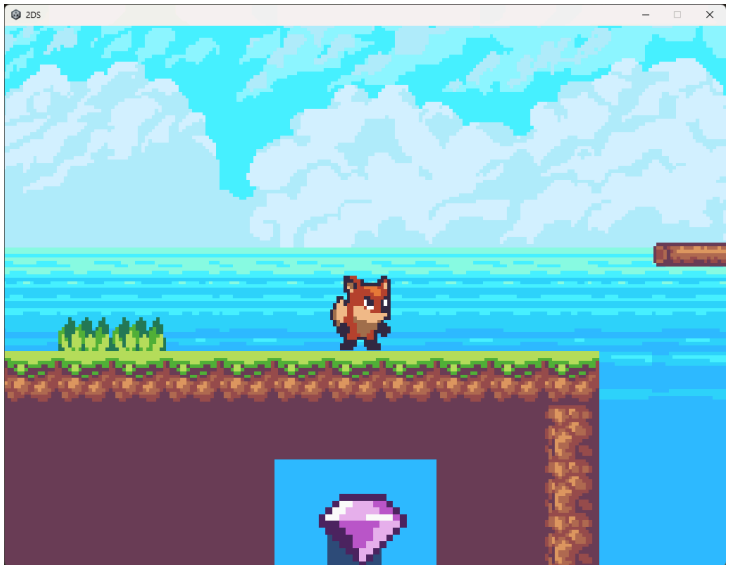
Challenge Name: SuperPatchBros'

Description:

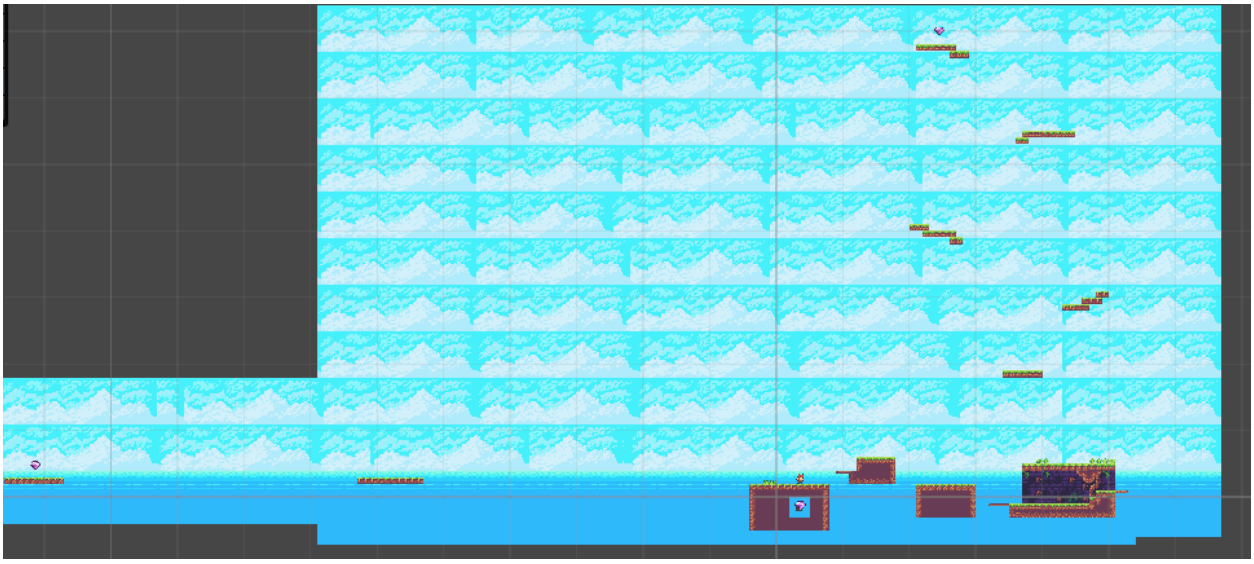
You're handed a Unity Windows build:

- Main EXE in your root export directory.
- DLLs like Assembly-CSharp.dll in <game_folder>*_Data\Managed\.

This is basically an Unity-based 2D platformer with three gems in the map, only one of which is clearly visible to the player, i.e. the one underground at spawn, with the location of the other two gems for the players to figure out.

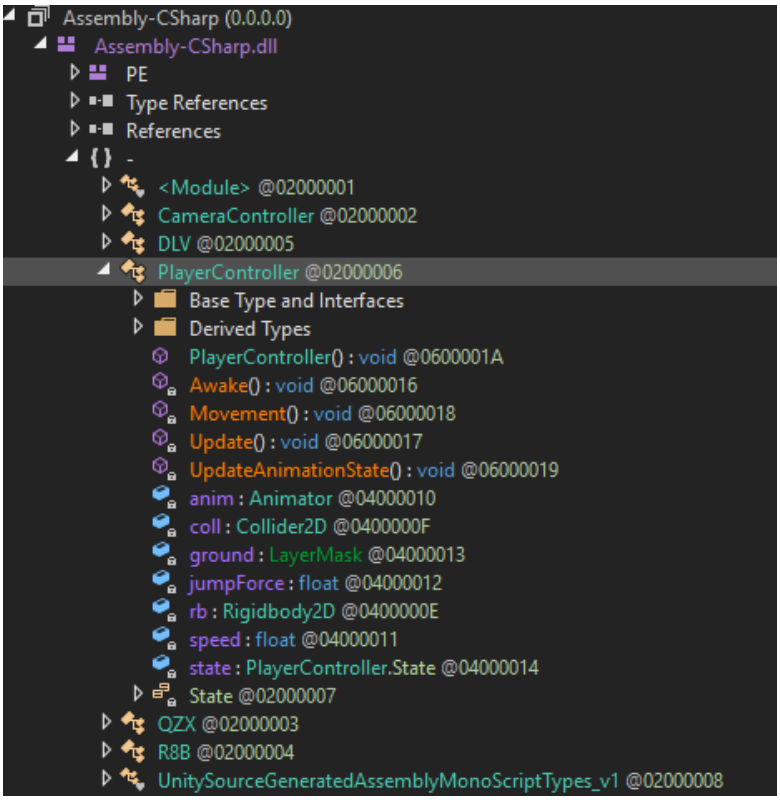


Here's the full map layout here from my Unity editor to give you, as the reader, a clear reference for where each gem is placed and how the game space is structured. Of course this isn't something players see in the challenge, but it's helpful context while reading the solution.



Unity games (unless the dev goes hard on obfuscation/stripping) are not native binaries. Instead of compiling everything to raw machine code in a .exe, Unity keeps pretty much all the gameplay logic in managed C# code as .NET assemblies. The big one is always Assembly-CSharp.dll inside *_Data\Managed. That DLL carries the actual player movement, collision handling, collectibles, even underlying UI logic for most games.

What this means: you're dealing with bytecode (IL), not pure assembly, which maps super cleanly back to readable C#. You'll find full class names, field names, method names. Unless there's paid obfuscation or super-aggressive stripping (think commercial packers), 95% of what matters is just sitting there for you to see.



Pop open Assembly-CSharp.dll in dnSpy. Scroll down your class list, PlayerController is the game's control center. You'll see everything you need: speed, jumpForce, movement logic, collider setup.
Movement happens in Movement() (RVA 0x000027FC, File Offset 0x000009FC), called every frame in Update(). Speed/jump are just fields:

```
// Token: 0x04000011 RID: 17
[SerializeField]
private float speed = 5f;

// Token: 0x04000012 RID: 18
[SerializeField]
private float jumpForce = 2f;
```

```
if (Input.GetButtonDown("Jump") && this.coll.IsTouchingLayers(this.ground))
```

And yeah, I will just be going through one of the possible methods, there are multiple patches and possible ways to go about this, creativity is part of the fun.

coming to the patches required for each gem,

Patch 1: Unlocking Gem 1 => Speed & Jump Boost

Addresses:

- Field Initializers:
  - speed (token RID 14, RVA location per dnSpy)
  - jumpForce (token RID 15, same logic)
- Awake():
  - Method RVA 0x00002401, File Offset 0x00000601

Edits:
Direct field editing is possible and would mostly work as well but for persistent override I would go with this way:

```
// Token: 0x06000016 RID: 22
private void Awake()
{
    this.rb = base.GetComponent<Rigidbody2D>();
    this.coll = base.GetComponent<Collider2D>();
    this.anim = base.GetComponent<Animator>();
    this.speed = 50f;
    this.jumpForce = 25f;
}
```

Recompile and save module. Now you'll easily reach Gem 1, at the leftmost section of the map.

Patch 2: Unlocking Gem 2 => Jump Boost

Honestly pretty much the first patch, tweak around with the speed and jumps and figure a way to discover and reach that gem. Actual bit for these were just exploration and figuring out the locations of the gems.

```
private void Awake()
{
    this.rb = base.GetComponent<Rigidbody2D>();
    this.coll = base.GetComponent<Collider2D>();
    this.anim = base.GetComponent<Animator>();
    this.speed = 10f;
    this.jumpForce = 25f;
}
```

Patch 3: Unlocking Gem 3 => Foreground Colliders

Addresses:

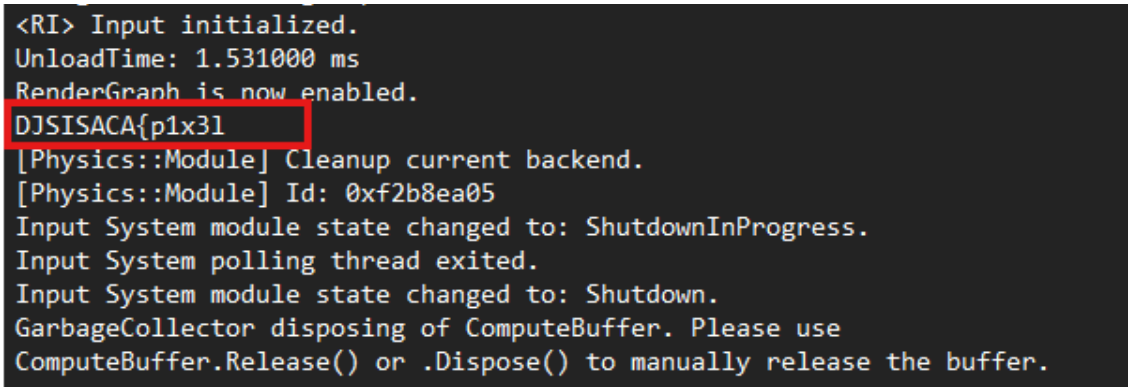
- Awake(): same as above
- Layer logic typically sits on Layer 3 (foreground platforms), you will have to do some trials to figure out the layer but this much is acceptable right.

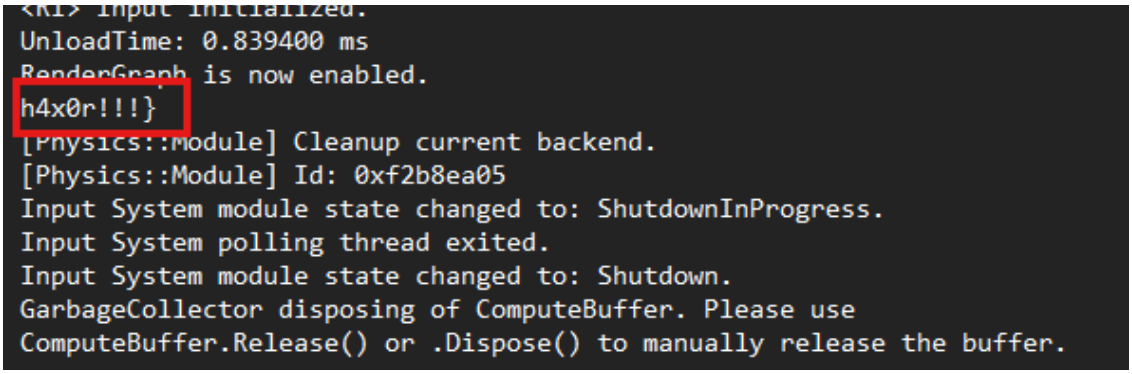
runtime patch in Awake():

```
private void Awake()
{
    this.rb = base.GetComponent<Rigidbody2D>();
    this.coll = base.GetComponent<Collider2D>();
    this.anim = base.GetComponent<Animator>();
    foreach (GameObject obj in UnityEngine.Object.FindObjectsOfType<GameObject>())
    {
        if (obj.layer == 3)
        {
            Collider2D col = obj.GetComponent<Collider2D>();
            if (col != null)
            {
                col.enabled = false;
            }
        }
    }
```

Recompile and save modules. All objects on layer 3 lose colliders, you'll fall straight through and get Gem 3. Common sense but if you try rendering out all collisions, you will phase through the gem as well.

And finally,
As mentioned in the description, every gem collected logs a message to
%appdata%\..\LocalLow\DefaultCompany\2DS\Player.log
Windows path expands to:
C:\Users\<your_username>\AppData\LocalLow\DefaultCompany\2DS\Player.log)





Flag: DJSISACA{p1x3l_sh3n4n1g4nz_h4x0r!!!}