CSE 3902: File I/O and Levels

Justin Holewinski

The Ohio State University

File I/O in C#

Common Types:

- `StreamReader`: Read string data from a stream/file
- `FileStream`: Read bytes from a file

```
using (StreamReader reader =
         new StreamReader(Path.Combine(Content.RootDirectory, "file.txt")))
{
    string line;
    while ((line = reader.ReadLine()) != null)
    {
        // Write the line we just read to stdout
        Console.WriteLine(line);
    }
}
```

Text Serialization Formats

JSON: JavaScript Object Notation

```
{
    "name": "foo",
    "values": [ 1, 2, 3 ]
}
```

YAML: YAML Ain't Markup Language

```
name: foo
values:
    - 1
    - 2
    - 3
```

XML: eXtensible Markup Language

```
<name>foo</name>
<values>
    <value>1</value>
    <value>2</value>
    <value>3</value>
</values>
```

CSV: Comma Separated Values

```
1,2,3,4,
5,6,7,8,
```

Lots of others...

Object Text Serialization in C#

C# (and .NET in general) has rich object text serialization support

- `System.Text.Json`
- `System.Xml.Serialization`
- `YamlDotNet.Serialization`

Primitive types, arrays, and structs of value types are supported by default

- `int`, `string`
- `int[]`, `string[]`
- `Vector2`, `Rectangle`
- Struct of primitive/value types

Most MonoGame reference types are not supported without significant work

- `Texture2D`, `SpriteBatch`

Design your serialization objects around structs of data

- Think of what you can easily represent with text
- Use texture names instead of actual `Texture2D` objects

JSON Serialization Example

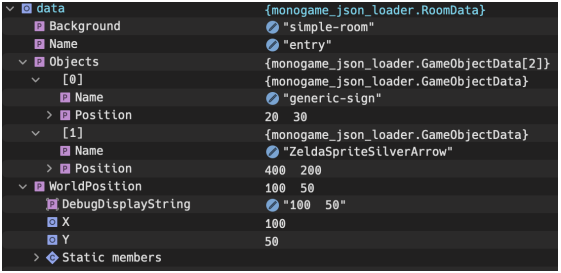
```
public class GameObjectData
{
    public Vector2 Position
        { get; set; }
    public string Name
        { get; set; }
}
```

```
public class RoomData
{
    public string Name
        { get; set; }
    public GameObjectData[] Objects
        { get; set; }
    public Vector2 WorldPosition
        { get; set; }
    public string Background
        { get; set; }
}
```

```
using (FileStream jsonStream =
    new FileStream(Path.Combine(Content.RootDirectory, "entry.json"), FileMode.Open))
{
    JsonSerializerOptions options = new JsonSerializerOptions {
        IncludeFields = true,
    };
    RoomData data = JsonSerializer.Deserialize<RoomData>(jsonStream, options);
    ParseRoomData(data);
}
```

JSON Serialization Example

```
{
  "Name": "entry",
  "WorldPosition": {
    "X": 100,
    "Y": 50
  },
  "Background": "simple-room",
  "Objects": [
    {
      "Position": { "X": 20, "Y": 30 },
      "Name": "generic-sign"
    },
    {
      "Position": { "X": 400, "Y": 200 },
      "Name": "ZeldaSpriteSilverArrow"
    }
  ]
}
```



C# Serialization Useful Links

JSON:

- https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/how-to
- https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/configure-options

XML:

- https://learn.microsoft.com/en-us/dotnet/standard/serialization/introducing-xml-serialization
- https://learn.microsoft.com/en-us/dotnet/api/system.xml.serialization.xmlserializer

Level Serialization

Sprint 3 will ask you to read level data from a file

General Process

- Choose a serialization format (JSON, XML, ...)
- Define what data is needed to fully represent a level
  - Rooms, enemies, doors, items
  - ???
- Define C# data structures that represent your serialized level data
  - Goal is to represent your entire level as C# structs containing only primitive types
- Design level as text in your serialization format
- Read level data on game load, read all needed content, and create object instances
  - Textures, sounds, enemy instances, player instance, item instances
  - ???