You can get a sample project here: https://github.com/Tichau/Schmup/tree/tp-datadriven if you don't want to use your project.

TP - Shoot'em up - Data-driven

PART 1 - DATA DRIVEN LEVEL DESIGN

Introduction : XML Serialization

Serialization is a mechanism for converting an object (such as an instance of a class, or a collection of objects) into a stream of bytes or characters that you can save to a file.

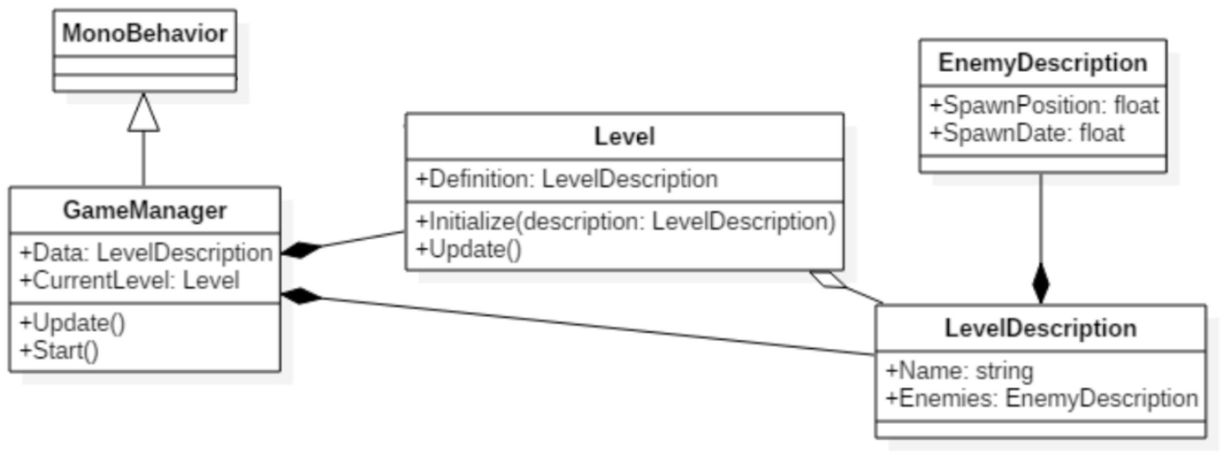
Many C# framework objects and classes can be serialized without adding any special directives or attributes to the code. By default, all public properties of a class are already serializable.

Example:

```
public class Test
{
	public String value1;
	public String value2;
}

Can be serialized in:

<Test>
	<value1>Value 1</value1>
	<value2>Value 2</value2>
</Test>
```

You can also control the serialization with c# attributes. For more details see here (or in the slides) : https://www.udemy.com/blog/csharp-serialize-to-xml/

The file XmlHelpers.cs contains the serialization/deserialization code.



Define your Data Structure

First we need to define a data structure. What data is needed to be externalized and how these data will be stored. You can find this data structure in the Scripts/Data folder of this project:
https://github.com/Tichau/Schmup/tree/tp-datadriven

You can also find a xml file with some data in the file Resources/Xml/Levels.xml

Question 1:

- **Load the level description with the function DeserializeDatabaseFromXML<LevelDescription>(myXmlFileAsset). Store them in the GameManager.**
- **Verify that your data are correctly loaded using breakpoints.**

Execute your level from your data

The game manager controls the execution of your game. It must load all the level descriptions from the XML and manage their execution.

Question 2: Modify your Game Manager to manage levels and progression.

- Game Manager deserialize all the level descriptions in the Start method and store them.
- We will now encapsulate all the logic of a level execution in a specific class
    - Create an instance of the Level component and initialize it with the LevelDescription (load method).
    - Execute the level following the given description:
      Check when and where to spawn the enemies in the Execute method.

```
public class Level {
    public void Load(LevelDescription levelDescription) {
        // Prepare a data structure to save which enemies you have/need to spawn (using
        // levelDescription)
        // Save the start time (using Time.time)
    }

    public void Execute() {
        // Check which enemies you have to spawn using the start time of your level
        // Foreach (enemy in EnemySpawn) {
        //    if (enemy.IsAlreadySpawn) continue;
        //    else if (enemy.NeedToBeSpawned(levelStartTime))
        //        Spawn(enemy);
        //}
    }
}
```

Question 3: When the level is finished, release it and start the next one.

- When the level is finished, call Release on the Level component.
    - To known that your level is finished you can use a timer (example: level 1 duration is 50 secondes)
    - It must release all the instances used (avatars, bullets, …). We need to have the ability to reuse instances for the next level.
- Continue the game by loading the next levels until the end of your data.

PART 2 - DATA DRIVEN LEVEL VISUALS

Add some scenery to your levels

You now have a game with data driven level design (you can define all the gameplay in an xml file to create new levels). But what about visuals, you will want to have some specific visual element in each levels so the player is not bored with your game visuals.

Separate your data using Unity scenes

Separate Game scene and Level scenes to have more control on the scope of your objects and to keep information between levels. The Game scene will take care of your persistent data/services. The level scenes will take care of elements having a life scope included in level lifetime (for example: level background decoration).
You can then ask graphists to directly modify level scenes without touching your game scene (it’s an example of separation).

Question 4: **Load a scene when you load a new level (and unload it at the end)**

- **Create a level scene with a specific scenery for the level 1.**
- **Add a reference to this scene in your level description (the scene name).**
- **Load your level scene on the top of the current one (see the code sample).**
    - Be careful the LoadSceneAsync operation is not instant, you need to wait the load completion before starting the level.
- **When the level is finished, unload the scene using SceneManager.UnloadSceneAsync method.**

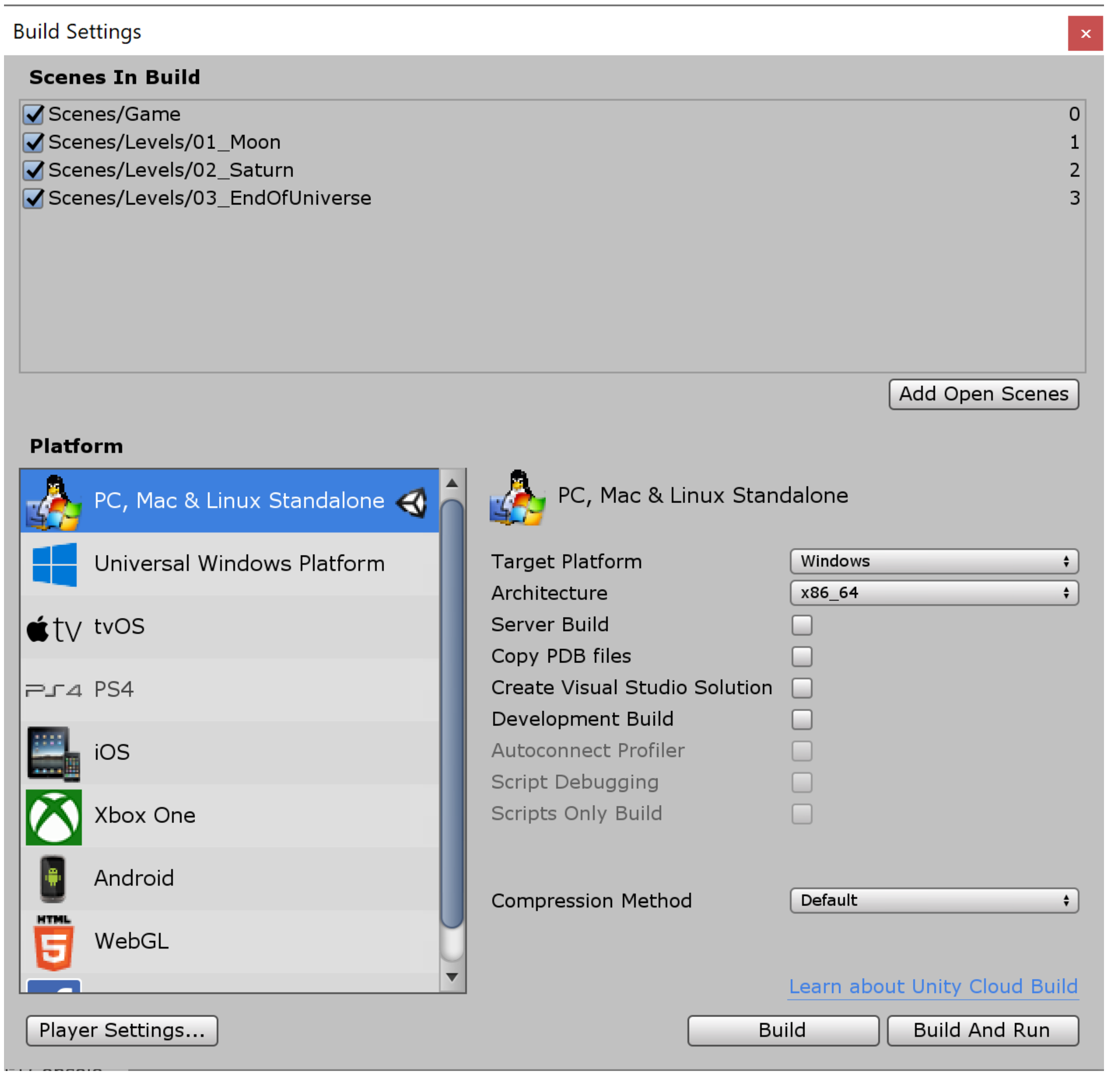
```
AsyncOperation loadSceneAsync = SceneManager.LoadSceneAsync(this.description.Scene,
LoadSceneMode.Additive);

while (!loadSceneAsync.isDone)
{
    // Wait for the level to be loaded.
    yield return null;
}
```

You can now ask a graphist to create some beautiful scenes to make your levels beautiful and ask some game designers to create new level designs in xml without writing a line of code.

Build your game

Question 5: **Build your game.**

Don't forget to add your scenes in the Build Settings (Ctrl+Shift+B)



Build Settings

Scenes In Build

✓ Scenes/Game	0
✓ Scenes/Levels/01_Moon	1
✓ Scenes/Levels/02_Saturn	2
✓ Scenes/Levels/03_EndOfUniverse	3

PC, Mac & Linux Standalone

Target Platform	Windows
Architecture	x86_64
Server Build	☐
Copy PDB files	☐
Create Visual Studio Solution	☐
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐
Scripts Only Build	☐
Compression Method	Default

You have now an executable of your game. You can send it to players!

Go further

Now, you can add more and more levels easily to your game. You can focus your development on other more interesting tasks such as:

- Create new enemies
- Add obstacles
- Create an editor to edit xml files.
- Create a collectible items system on the same scheme that the enemies …
- … and instancing it by xml in your levels !
- Create a notion of enemies pattern to facilitate level design. (EnemyPatternDescription that contains enemies, LevelDescription can contains EnemyPatternDescriptions).
- and more and more features ...