# Scenario 2:

You are coding the gameplay of a cool new 3D arena shooter that has a design requirement involving many enemies, bullets and coins to be on screen at once. You notice when initially testing your game the framerate is uneven and the game seems to even freeze at times.

---

- Questions:

1. **What steps do you take to deduce the problem?**
   - *The first thing we need to do is to take a look at the Profiler, look at the GC Allocation, as well as the calls, and scripts to find what could be causing the drop of frame rate, after finding out the issue (in this type of scenario is likely to be a GC allocation issue), start working on fixing it.*

2. **What are some programmatic solutions you could propose to fix the problem?**
   - *One of the things that can happen when we have a lot of bullets on the screen is that we may be having a GC allocation issue, that is that when instantiating and eventually destroying the bullets we call the garbage collector and the more bullets we have, the more of a performance hit our game will have (I'm assuming here that the enemies and coins will have the same issue since we'll be spawning them a lot), so one of the solutions would be **Object Pooling.** Object Pooling is a design pattern used in games where we instantiate at the beginning of the game all those objects that will be using a lot, and instead of destroying them, we return them to the pool so we can later reuse them.*
   - *Other thing we can check on the enemies, is the polygon count, depending on the type of game and the angle of the camera we could use LOD's to have the enemies far away in a low-poly version and as the enemy gets closer to the player we swap the low-poly models for a high-poly one.*
   - *Other solution could be occlusion culling, so whatever we are not looking at right now we can deactivate the renderer component.*

3. **How will you confirm your changes had a positive impact?**
   - By checking again the profiler.
   - Checking the "Stats" by clicking the "Stats" button on the game tab, so we can take a look at the batches, tris, frame rate, etc (note that the frame rate that is shown in the "Stats" and the Profiler is not the actual frame rate that the player will have on the target device, since we have to take into consideration the Unity Editor overhead, for a more accurate profiling we can enable the standalone profiler).

# Scenario 3:

You're deep in the middle of making what will surely be the greatest ARCore based 3D sticker app the world has ever seen. At the morning stand up you hear that the sound designer has made a pass updating the existing UI sound files while the art team have submitted 10 new 3D sticker prefabs bringing the total to 250! Excited to see the new changes you rush back to your

desk and update. After making a new build you notice the size of the .apk has doubled in size, far too big for the changes mentioned!

---

## • Questions:

1. **How do you go about figuring out the cause of the increase?**
    - Since this is a file size issue, we go to the Editor log (https://docs.unity3d.com/Manual/ReducingFilesize.html) and see what is taking that much space in our build and which assets are candidate for optimization.

2. **How will you inform the team of your findings?**
    - I will share the Editor Log file of the build and go to the part of the "Build Report" where it shows the use of the assets from the resources folder as well as the other type of assets (also the space and percentage they take).

3. **What steps do you take to reduce the size?**
    - In this particular example I'll take a look at the audio files and if they are *.mp3 files I would suggest to switch if for *.ogg files, since that format Is more compressed and reduced in size (.ogg is the open source equivalent of the mp3 but smaller in size, since the algorithm used to compress it).
    - For the 3D sticker prefabs I would take a look at the poly count of the 3D model of the sticker and see if we can reduce it using a model tool such as Blender. Also for the textures we would need to compress the textures, even though Unity already does that, we can specify per platform; and also to make sure that the texture is properly compressed and optimized, we need to make it POT (Power of Two) since GPU's tend to work better with those values.
    - Specially if working with mobile platforms (like in this case) build files are extremely important due to AppStore and Play Store build size limitations. So if those sticker prefabs are absolutely needed, we could use Addressables (previously Asset Bundles) to deliver content remotely after the initial install over a CDN or from a GIT Repository.
    - Other option is to reuse textures for multiple asset if possible (like 2 prefabs having the same background, instead of adding the texture twice, make them share it).