Scenario 1:

You are brought on to help with the latest LFL VR masterpiece that has already been in development for 6 months. Your support will help take it from a working alpha to a feature complete beta. You are familiar with the functionality as you have played around with the experience casually and even understand the overall structure from a few offsite lunches with the lead engineer. After being assigned a few "simple" bugs you get to work. Immediately, you find a core piece of code that can be improved for efficiency is being used throughout the project and has been in the code base for almost 5 months.

Question 1: How would you attempt to understand the design of a large code base in a short period of time?

My first option is always to reach out. As a new employee, no matter how good I am, it will take a lot of time and work to get my head around the overall structure of the codebase. If I'm asked to find a bug affecting a certain part of the game, I will ask my supervisor and colleagues what code files contain the functionality I'm looking for.

Once I'm looking at individual files, I'll collapse all the functions and read their names and lists of arguments. My entire strategy is to reduce the amount of information I need to actually understand. When joining a new project, the information I feel like I have to understand is usually all of the information. In a codebase with millions of lines of code, this is an impossible task, so from moment one, my focus is to narrow the scope of what I have to learn. Most developers and engineers are good enough to write function names that are pretty coherent and understandable.

Only once I've looked through the file and picked out a couple functions that seem relevant will I actually start trying to read any code. Even then, if the bug I'm looking for has to do with artificial intelligence, I'll skip over code if I know it's unrelated.

Once I'm actually looking at code, the process gets a little more boring and messier. From this point, I'll try to understand control flow as one function calls another, which talks to another file and set of systems. By this point, things like keyboard shortcuts like "find all references" and "go to original" start becoming very helpful to me as I track down how this piece of functionality works. By this point, hopefully, I've reduced a task that would have required understanding the project in its entirety, to a task that just requires reading typically a couple dozen lines or so.

Question 2: What steps will you take to understand the ramifications of the potential refactor?

This answer will be surprisingly like the first. My instinct right away would be to ask. If someone already knows the answer, I can narrow things down incredibly for myself. I'm not ashamed at all to admit when I don't have the information on hand to solve a problem. However, given that others might not have exact answers or don't have the time to go through it all, I revert back to keyboard shortcuts to jump around. Here in particular, pulling up a list of all instances where a method is called will be particularly useful.

Question 3: How do you balance the task of fixing the bug with the investigation?

Balancing different priorities is a big skill for developers, but I would prefer not to have to divert my attention at all. I would immediately notify my supervisor of the issue I found. If I was put on a task, the expectation given to me is usually that I complete that task. If I discover something, it can be catalogued and possibly be added as a new task in and of itself. It's more advantageous to the company this way as they will be given the opportunity to respond to discovery and myself as I can now allocate my time properly to both tasks.

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.

Question 1: What steps do you take to deduce the problem?

Performance tools. If I'm using an engine that has a performance monitoring toolset, I'd happily jump to use that. Otherwise, there are third party performance capture applications that also work great. Once I get a readout of how long each frame is taking to render and specifically which functions are causing slowdown, I'll have an exact answer without the need for any guesswork.

Assuming for any reason that I either didn't have any of these tools on-hand or couldn't use them, my first guess would be to pull up the code for spawning in the enemies and objects on screen and see if they're being handled smartly or not.

Question 2: What are some programmatic solutions you could propose to fix the problem?

Object pooling is my first thought, especially with things like coins and enemies. Instantiating things into your game world can take a very long time, especially if for any reason, they aren't stored in system memory. By spawning a large pool of both of coins and enemies on load and deactivating them when they're not needed anymore instead of deleting them from memory, whenever I need the game to spawn new instances, I can just take my deactivated copies, move them to the right spot, then turn them back on.

If you're having rendering issues in particular, while this is a very standard feature, it's worth seeing if you currently have frustum culling in the project. This would stop anything outside the view of the camera from rendering.

Lastly, it might also be worth looking into seeing if you can move some of the logic for these things over into shaders. The coins in particular might be a good fit for this. By offloading the processing to the GPU instead of CPU, you could potentially save a lot of time per frame by taking

advantage of the parallel computing power of a dedicated graphics card. This one's a little tricky though, as a poor implementation could actually make framerates even worse if the shader has to refer back to the CPU too often.

Question 3: How will you confirm your changes had a positive impact?

Again, the most obvious answer here to me would be performance tools. I could compare how much time per frame was spent running certain functions before and after my changes, as well as overall frame time.