Question 1

During my time at ADP, a company I still hold very close to my heart, I worked on a large-scale global operation with teams across Brazil, North America, India, and EMEA. We had monthly releases, and coordinating testing across all these regions *was a huge effort.* At that time, we used Java + RestAssured for backend automation, and I was responsible for maintaining and adding new test cases alongside development. Since all regions followed this same stack, the volume of automated tests was massive.

One day, our lead architect proposed migrating all backend testing to Groovy + Cucumber, believing it would make it faster and the maintenance easier. His opinion carried a lot of weight, and I truly respected his technical expertise. But before saying anything, I took some time to think carefully about the impact.

Even though the idea wasn't necessarily bad, I realized the cost and effort of this migration would be huge:

- We would need to rewrite hundreds of existing tests.
- Entire global teams would have to learn a new language and framework from scratch.
- The migration could (and would!) easily delay releases and create inconsistencies between regions.

I had prior experience with Groovy and Cucumber, so my concerns weren't about comfort zones: they were about feasibility and scalability. I decided to share my perspective respectfully, explaining:

- The risks of the migration considering the size of our testing suite.
- The learning curve for teams worldwide.
- The impact on timelines and coordination across all regions.

In the end, after a discussion, we agreed to stick with Java + RestAssured and instead focus on incremental improvements to make our existing framework more maintainable.

This experience taught me a lot about balancing respect and practicality. I try to separate personal preferences from what's objectively better for the project, always aiming to make decisions that support the team as a whole.

Question 2

I haven't worked directly with testing a game engine, but at ADP (again!)I worked on testing an internal SDK that handled Single Sign-On (SSO) and integrated multiple ADP products into a unified authentication flow. Since this SDK wasn't exposed directly to end users, the testing focus was very different. Instead of mobile automation tools like Appium, we concentrated on API-level and integration testing to validate the authentication logic.

I designed the test plan in three main layers:

- Unit Coverage *(handled by devs)* → Token generation, encryption, and expiration logic.

- Integration Testing *(my main focus)* → Using Java + RestAssured, I simulated authentication requests, validated responses, checked error handling, and ensured proper token exchanges between products.

- E2E Flows → Verified that authentication worked seamlessly across multiple ADP applications.

The biggest challenge was the real-time nature of authentication across different regions: we had to coordinate multiple environments and manage dependencies between products. So, this experience taught me how to design a scalable, efficient test strategy for shared SDKs, focusing on reusability, API-first validation, and cross-product integration while keeping the process maintainable.