# Development Testing

Consistent with Test Driven Development, we decided on testing strategies for the testable systems in our architecture diagram. These testing frameworks were decided on:

- **Front End UI** - APK Build test
- **Graphics Rendering** - Visual Testing
- **.CSV Level Handler** - APK Build Testing
- **Physics Engine** - JUnit Testing

**APK Build Testing**

Android studio provided an **APK Build** feature which allowed us to test things like functionality of certain *non-numeric* features, as well as robustness of integrated systems. This would compile the project state and export it as a testable APK which installed onto a connected mobile device (a terminal would display system messages describing the events occuring and processes running during application usage). This made more sense for complex integrated systems like the **.CSV Level Handler** and **Front End UI**. We were also able to test the usability factor of the application (which is importatant when a game is concerned).

**Visual Testing**

Earlier on in development, where the 3D assets were not ready, we used visual placeholders in order to ensure the correctness of the renderer during the **APK Build Tests**. Through this, we were able to test that objects created in out graphical engine would be rendered in the correct position, orientation and size.

**JUnit Testing**

Testing the correctness of methods in both normal and edge through the JUnit Testing framework proved the robustness of systems and methods. In back-end systems like our **Physics Engine**, which was implemented with independently functioning methods, it was the only system that made sense to use this kind of testing. Below is an example of some **JUnit** test cases used to test the **Physics Engine**:

## Testing Table Sample:

| Test | Testing Condition | Pass/Fail |
|---|---|---|
| @Test<br>forceAppliedTest | `//Set the comparison parameters`<br>`expectedAcc = (240f, 150f, 0f);`<br>`//Set context`<br>`result = forceApplied(initAcc, inputForce,`<br>`movement.getMass(), movement.frametime);`<br>`assertEquals(expectedAcc, result, delta);` | **Pass** |

| Test | Testing Condition | Pass/Fail |
| --- | --- | --- |
| @Test<br>CalcVelTest | //Set Comparison Parameters<br>inputAcc = (240f, 150f,0f)<br>expectedVel = (8f, 5f, -6f);<br>//Set Context<br>result = movement.calcVel(movement.getVel(), inputAcc,<br>movement.frametime);<br>assertEquals(expectedVel, result, delta); | **Pass** |
| @Test<br>CollisionTrueTest | //Set context<br>InitPos = (2543f, 3500f,500f );<br>movement.setPos(initPos);<br>movement.setMovementSize(droneObject);<br>obj = ApartmentsObject(4000f, 660f, 4000f, 1, 1, 1);<br>movement.isCollision(movement, obj);<br>assertTrue(movement.collided); | **Pass** |
| @Test<br>UpdateMovement<br>Test | //Comparison parameters<br>expectedAcc = (240f, 150f, 0f);<br>expectedVel = (6f, 3f, -4f);<br>expectedPos = (0.27f, 0.17f, 0f);<br>//Set context<br>movement.updateMover(expectedAcc, expectedVel,<br>expectedPos, movement);<br>assertEquals(expectedAcc, (movement.getAcc()), delta);<br>assertEquals(expectedVel, (movement.getVel()), delta);<br>assertEquals(expectedPos, (movement.getPos()), delta); | **Pass** |