1. Programming Languages (5 points)

C#: The programming language we are using for our game project is C#. We use it in custom scripts to control game objects, such as handling character movement, camera behavior, and enemy AI. We chose C# because it is the only scripting language supported by Unity, and it integrates well with the engine's built-in tools. It also makes it easier to organize our code and manage game mechanics efficiently.

2. Platforms, APIs, Databases, and other technologies used (5 points)

Unity: We use Unity as our primary development platform to handle rendering, physics, animations, and scene management. It provides built-in tools that simplify game development and allows for quick iteration.

Visual Studio Code/VisualStudio Community: We use Visual Studio Code and Visual Studio Community as our code editors for writing and debugging C# scripts. It offers useful features like IntelliSense, extensions, and Git integration, making development more efficient.

Unity Asset Store: We use the Unity Asset Store to find and integrate premade and free assets to use in our game. This allows us to spend our time building and programming game mechanics rather than making art.

Unity's Physics Engine: Used to handle collisions, gravity, and movement mechanics, ensuring realistic interactions in the game.

Tilemaps (For Level Design): We use Unity's Tilemap system to efficiently create game environments with reusable assets.

Version Control (Git): We use Git to track changes in our project, allowing for collaboration and version history management.

3. Execution-based Functional Testing (10 points)

*Describe how/if you performed functional testing for your project (i.e., tested for the **functional requirements** listed in your RD).*

Unity provides the ability to run the game within Unity itself, as well as assigning objects directly to script attributes. This allows us to test functional requirements in the game by testing them live. For example, The in-game combat system has several attributes that must be assigned at the beginning of the battle, including a player, enemy, and so on. Unity allows you to manually set these attributes in the Inspector panel, allowing us to easily test the battlesystem with dummy enemies. Once we were ready to connect the battle system to a real enemy encounter, we ran the game in Unity in the overworld, interacted with a dummy enemy we set in one of the dungeons, and played through a battle to check for correct functionality.

Each person on the team conducts this testing on their own machine to ensure all functionality works as intended and nothing is missed. This also ensures that the code existing in Github is orderly and as intended.

4. Execution-based Non-Functional Testing (10 points)

*Describe how/if you performed non-functional testing for your project (i.e., tested for the **non-functional requirements** listed in your RD).*

For smooth and intuitive movement mechanics for the primary player, we have implemented collision scripts that implement intended interaction with objects and no jitters with movement. Through simulated

gameplay, we have verified that player movement mechanics were responsive to user keyboard input and that objects, such as trees, trigger a collision and that scene changes, such as entering the dungeon doors, are indicative of a smooth clean change in the correct orientation. Both Windows and MacOS systems have been compatible with the Unity and gameplay software.
The user interface depicts minimal loading time and is displayed as intuitive and easy to navigate, such as the clear battle system displaying clear buttons for ATTACK and SKILLS. We have demonstrated these non-functional requirements through gameplay on several devices of our current map and interface.

5. Non-Execution-based Testing (10 points)

Describe how/if you performed non-execution-based testing (such as code reviews/inspections/walkthroughs).

Non- Execution based testing was performed through active gameplay for code review as we test and debug the scripts for camera tracking, battle system, movement controls, player navigation, collisions, and scene changing. This was conducted as a group demonstration, and on individual machines for testing. Version control was also maintained through a GitHub Repository for Increment 2, ensuring every member has the correct version and access to revision history of debugging and newly introduced commits. Our documentation has aligned with our gameplay experience, demonstrating proper implementation of functional and non-functional requirements.