# **TECHNICAL REQUIREMENT:**

Game development involves a wide range of technical requirements, depending on the complexity and platform of the game. Here are some common technical requirements and considerations in game development. These technical requirements can vary based on the type of game (e.g., 2D, 3D, mobile, console, PC) and the specific needs of your project. It's important to plan and document these requirements thoroughly before starting game development to ensure a smoother and more successful development process.

1. **Game Engine:** Select a game engine that suits your project's needs. Popular game engines include Unity, Unreal Engine, Godot, and others. The choice of engine will influence the tools and programming languages you use.

2. **Programming Languages:** Depending on the game engine, you may need to use specific programming languages. For example, C# is commonly used with Unity, while Unreal Engine primarily uses C++.

3. **Graphics and Art**: Create or source 2D and 3D assets for your game, including characters, objects, textures, and animations. You may need software like Adobe Photoshop, Blender, Maya, or 3ds Max.

4. **Sound and Music:** Develop or obtain audio assets, including background music, sound effects, and voiceovers. Tools like Adobe Audition, Audacity, or other digital audio workstations (DAWs) may be needed.

5. **Physics and Simulation:** Implement physics engines and simulation systems for realistic interactions within the game world.

6. **User Interface (UI):** Design and implement the game's user interface, including menus, HUD, and in-game displays.

7. **Game Logic and Scripting**: Write code to handle game logic, such as player controls, AI behavior, game rules, and events.

8. **Networking:** If your game is multiplayer, you'll need to implement networking functionality to support online gameplay. This includes server-client communication and synchronization.

9. **Optimization:** Optimize the game's performance to run smoothly on the target platform. This may involve reducing polygon counts, implementing level-of-detail (LOD) systems, and more.

10. **Testing and Debugging:** Use debugging tools and conduct thorough testing to identify and fix bugs and glitches.

11. **Platform-specific Requirements:** Consider the technical specifications and guidelines for the platforms you're targeting, such as PC, consoles, mobile devices, or virtual reality.

12. **Distribution and Deployment:** Prepare your game for distribution, whether through app stores, digital distribution platforms (Steam, Epic Games Store), or other methods.

13. **Security:** Protect your game from cheating, hacking, and unauthorized access, especially in multiplayer games.

14. **Performance Analysis**: Monitor and analyze the game's performance using profiling tools to identify bottlenecks and optimize performance.

15. **Cross-Platform Compatibility:** If you plan to release your game on multiple platforms, ensure it works well on each of them.

16. **Documentation:** Maintain comprehensive documentation for your code and game design, making it easier for the development team to collaborate and for others to understand and modify the game.

17. **Version Control:** Use version control systems (e.g., Git) to track changes, collaborate with a team, and manage different versions of the game.

18. **Legal and Licensing:** Ensure compliance with copyright and licensing requirements for any third-party assets used in the game.

19. **Accessibility:** Consider accessibility features to make the game playable by a wider audience, including those with disabilities.

20. **User Experience (UX) Design:** Incorporate good UX principles to make the game intuitive and enjoyable to play.

# DEVELOPMENT TIMELINE:

Week 1: Concept and Design

**Week 1,** Define the game concept, genre, and core gameplay mechanics.

**Week 1:** Create a design document, including game mechanics, level layouts, and art style.

**Week 1:** Begin asset creation or gather pre-existing assets if available.

**Week 1,** Initial design and concept review.

**Week 2:** Prototyping

**Week 2,** Create a basic prototype with essential gameplay elements to test your ideas.

**Week 3:** Development

**Week 3:** Start building the game's core functionality, including player controls, game mechanics, and basic levels.

**Week 3,** Art and Assets

**Week 4:** Continue creating or collecting assets, including 2D/3D models, textures, and sound effects.

**Week 4:** Level Design

**Week 5:** Design and create additional levels, balancing gameplay and difficulty.

**Week 5:** Refinement and Bug Fixing

**Week 5,** Polish the game, fix bugs, optimize performance, and improve user experience.

**Week 6**: Playtesting

**Week 6:** Conduct playtesting and gather feedback from players to make necessary adjustments.

**Week 6,** Finalization and Testing

**Week 7,** Finalize the game, perform rigorous testing, and prepare for release.

**Week 7:** Submission

# 3D OBJECTS AND SCENE MANAGEMENT:

Creating a complete game scene in Unity involves many detailed steps and a lot of code, which I can't provide in a single response. However, I can give you an overview of some key components and scripts you might need for your two stages: the basic stage and the boss stage. Remember that you'll also need to create 3D models, textures, and animations for your characters and objects. Unity provides a wide range of tools and resources for these tasks. It's crucial to break down the development into smaller tasks, plan your game's architecture, and gradually build and test each component.

**Stage 1 (Basic Stage):**

1. **Player Movement Script:**

- Create a C# script to control the player's movement using Unity's Input system.

- Implement functions for moving, jumping, and attacking with the "Sharp Knife."

2. **Enemy AI Script**:

- Create an enemy AI script that defines enemy behavior, such as patrolling, following the player, and attacking.

- Implement health and damage systems for enemies.

3. **Coin Collection Script**:

- Write a script to handle coin collection when the player character collides with coins.

- Keep track of the player's score using a score manager script.

4. **Scene Management:**

- Implement a script to handle the transition to the next stage when the player reaches the win condition.

- Use Unity's Scene Manager to load the next scene.

**Stage 2 (Boss Stage):**

1. **Player Upgrade System:**

- Create a shop or upgrade system where the player can spend the coins collected in Stage 1 to purchase weapon upgrades.

- Implement logic for upgrading the player's weapon.

2.**Boss AI Script:**

- Develop a script for the boss's behavior, including movement patterns and attack patterns.

- Implement health and damage systems for the boss character.

3. **Key and Partner Rescue Script:**

- Create an object for the key and a script to handle key collection.

- Implement a mechanism for freeing the partner character once the key is collected.

4. **Game Completion Script:**

- Write a script to trigger a victory screen when the boss is defeated, and the partner is rescued.

5. **Game Over Script:**

- Implement a game over condition if the player's health runs out during the boss battle.

6. **UI Elements:**

- Design and implement UI elements such as health bars for the player and boss, an inventory for upgrades, and on-screen prompts.

7. **Sound and Music:**

- Add audio assets and use Unity's Audio Source component to manage background music, sound effects, and voiceovers.

8. **Animations:**

- Create animations for character movements, attacks, and interactions like opening the partner's cage.