System Requirements Document

By: Nathaniel Madrigal, Alexander Madrigal, Wyatt Bender,
Lance Heinrich, Matias Avila, and Chester Lee

Functional Requirements:

<u>User Interface</u>

Title Screen

- **Description**: The title screen is a scene that allows users to navigate between playing the game, selecting game modes (single-player or multiplayer), accessing save data, accessing game options, accessing account information, accessing friends, and closing the application. A global leaderboard displaying the highest scoring runs will also be displayed to the user.
- **Priority Level**: High
- **Possible Team Members**: Nathaniel, Chester
- **Inputs**: Player inputs (keyboard, mouse, gamepad), User selection (game mode, settings, leaderboard).
- **Outputs**: Navigational feedback, menu transitions, leaderboard display
- **Processes**:
    1. The title screen scene will be the first scene entered when executing application
    2. A leaderboard is displayed onto the screen listing information saved in from runs with the highest scores sorted greatest to least
    3. A list of options are displayed and available for the user to select: “start game”, “saves”, “options”, “account”, and “quit game”
    4. **Start game:** The user can select between single-player, local multiplayer, or online multiplayer. Selecting single-player will move the user to the starting game scene. Selecting local multiplayer will display a menu waiting for two player inputs to be connected. After the connection between two controllers is recognized, users are moved to the starting game scene. Selecting online multiplayer will display a menu to invite or join a friend to play in the same session before switching to the starting game scene.
    5. **Saves:** Users can select between multiple saves. Starting a game will use the progress of the currently selected save and updates the currently selected save with.
    6. **Options:** A list of game settings can be changed including master, music, and sound effects sliders and controller input settings.
    7. **Account:** The user can change profile name, add and remove friends, log out of current account, or login to an account

8. **Quit game:** Closes the application.

- **Dependencies:** Requires access to leaderboard data, save data, and account data

Pause Menu

- **Description:** The pause menu shall provide users options such as resume game, settings menu, save and exit to the title screen. Within the menu, an interface shall display the upgrades taken in a run and it should support local and online multiplayer where an alternative interface will show upgrades taken for multiple players. In single-player and local multiplayer mode, the game time shall be paused where all game objects will be held in place. In online multiplayer mode, each player can pause independently, but only the player who pauses will see the pause menu while the other player can continue playing. In this scenario, the game time is not paused and the paused player might receive damage. Receiving damage will interrupt the pause menu and return to the gameplay interface.
- **Priority Level:** High
- **Possible Team Members:** Nathaniel, Chester
- **Inputs:** Player inputs (keyboard, mouse, gamepad)
- **Outputs:** Player controls disabled, game world continues running (online multiplayer), interface for viewing upgrades taken during the run, menu options (restart run, save and exit, settings)
- **Process:**
    - Player presses pause button and pause menu appears
    - In single player and local multiplayer, the game gets paused (all objects and timers frozen).
    - In online multiplayer, only the pausing player's controls are disabled and they see the menu while the other player can continue playing
    -
- **Dependencies:**
    - Player upgrades, multiplayer gamemode

Map Menu

- **Description:** The map menu shall display an interface which will show level progression and room location where the player can locate the room through gameplay interface. A minimap will be displayed on the gameplay interface which shows a layout of the surrounding environment with limited range. The player can access a bigger map layout through the map menu. The menu shall provide details such as the room types (exploration room, boss room), icons for interactable NPC (shopkeeper, quest giver), and waypoint to certain important events.
- **Priority Level:** Medium
- **Possible Team Members:** Nathaniel, Chester, Wyatt

- **Inputs:** Player inputs (keyboard, mouse, gamepad), Player coordinate location
- **Outputs:** Player controls disabled, game world continues running (online multiplayer), interface for viewing map and level progression during the run
- **Process:**
    - Players can access the map menu by clicking on the minimap in the gameplay interface or pressing on the shortcut key.
    - In single player and local multiplayer, the game gets paused when the menu is called (all objects and timers frozen).
    - In online multiplayer, only the pausing player's controls are disabled and they see the menu while the other player can continue playing.
    - The player can zoom in and out the map, drag the map around and interact with icons and waypoints.
    - The player can view an overview of the level progression in a different interface where the level will branch out to several layers of levels and end in a final boss level.
- **Dependencies:** Gameplay interface, level generation system, player position system

Gameplay Interface

- **Description:** The gameplay interface provides real-time information to the player, including health, currency, skills, cooldowns, and status effects. It dynamically updates as the player progresses throughout the game. The gameplay interface shall display the camera which follows the character whenever the player moves or attacks.
- **Priority Level:** High
- **Possible Team Members:** Nathaniel, Chester
- **Inputs:** Player Stats
- **Outputs:** Health bar, currency display, cooldowns, status effects
- **Process:**
    - The game continuously tracks the player's stats including health, currency, skills, and active status effects.
    - The data is dynamically fed into the HUD
    - Health bar adjust in real-time as damage is taken or health is regained
    - Currency display is updated whenever Player collision with the currency occurs
    - Skill cooldown timers are displayed for each ability that is active
- **Dependencies:** Player stats

<u>Player</u>

Player Movement

- **Description:** Players shall be able to move vertically and horizontally within levels. Players shall be able to walk horizontally across platforms, move horizontally in the air, jump on the ground and in the air, vertically climb on climbable platforms, and go through drop down platforms.

- **Priority Level:** High
- **Possible Team Members:** Alex, Lance, Wyatt
- **Inputs:** Player inputs (keyboard, mouse, gamepad)
- **Outputs:** Change in player location and player animation
- **Process:**
    - When players collide with normal platforms, players can no longer move in the direction of that platform.
    - When players input a horizontal direction on the ground or in the air, players will accelerate towards that direction until a final velocity is reached and will decelerate to zero when run is not imputed.
    - Players shall be able to input jump when on any platform, and they shall be able to input jump in the air once until players touch the ground again. Jumps shall be vertical impulses.
    - While players collide with climbable platforms and input movement vertically, they shall move in that vertical direction without being able to run.
    - If players are on top of a drop down platform and input jump, the collision between the player and drop down platform shall be removed until the player is no longer colliding with the drop down platform.
    - When players are airborne, players will accelerate toward the down direction with constant gravitational acceleration.
- **Dependencies:** Playable characters, collision system

Player Interact

- **Description:** Players shall be able to interact with entities within levels.
- **Priority Level:** High
- **Possible Team Members:** Alex, Lance
- **Inputs:** Player inputs (keyboard, mouse, gamepad)
- **Outputs:** Changes to player health, currency, and upgrades
- **Process:**
    - When players collide with an entity and input interact, the game shall delete that entity. Entities shall have unique graphics and cause different results depending on the entity interacted with. Entities shall heal players, give or use players currency, and give player upgrades.
- **Dependencies:** Health, currency, player upgrades

Player Evade

- **Description:** Players shall be able to input evade which temporarily restricts horizontal movement, forces players to temporarily move in the horizontal direction that player is currently facing, and temporarily gain invincibility.
- **Priority Level:** High
- **Possible Team Members:** Alex, Lance
- **Inputs:** Player inputs (keyboard, mouse, gamepad)
- **Outputs:** Change in player location and player animation
- **Process:**
    - When players input evade, horizontal movement is disabled. The player moves in the horizontal direction they are currently facing for a short period of time where they are invulnerable from taking any damage.
- **Dependencies:** Playable characters, collision system (for perfect evade)

Player Attack

- **Description:** Players shall be able to input attacks which create either a melee or projectile hitbox which deal damage to enemy NPCs during that period of time. Attacks are comboable and loop. Players shall have different attacks depending on the character that they are controlling.
- **Priority Level:** High
- **Possible Team Members:** Alex, Lance
- **Inputs:** Player inputs (keyboard, mouse, gamepad)
- **Outputs:** Change in player animation and enemy NPCs stats
- **Process:**
    - Depending on the character selected, attacks can have different hitboxes, deal different damage, deal different stun, and different smaller attacks.
    - Attacks shall be broken up into a fixed number of smaller attacks. Inputting attack causes smaller attacks to occur in fixed order until all smaller attacks are dealt, and the next inputted attack shall be the first small attack.
    - If the attack is melee, it shall create a hitbox on the player which deals damage to enemy NPCs for the duration of the attack.
    - If the attack is projectile, it shall create an object with a hitbox which deals damage to enemy NPCs if the enemies collide with the object. Upon creation, the object should have an impulse in the direction that the player is facing. The object should be destroyed once it collides with enemy NPCs or collides with a platform.
- **Dependencies:** Playable characters, collision system, enemies stats

Player Skill

- **Description:** Players shall be able to input skills which deal damage to enemy NPCs and are unique to the character controlled. Once inputted, skills shall have timer cooldowns.

- **Priority Level:** High
- **Possible Team Members:** Alex, Wyatt, Lance
- **Inputs:** Player inputs (keyboard, mouse, gamepad)
- **Outputs:** Change in player animation and enemy NPCs stats
- **Process:**
    - Depending on the character selected, skills can have different hitboxes, deal different damage, deal different stun, and have different cooldowns
    - If a skill is not on cooldown and is inputted, that skill shall be used causing the timer cooldown to start. Until the end of the cooldown, that skill cannot be input by players.
- **Dependencies:** Playable characters, collision system, enemies stats

Gameplay

Health

- **Description:** Health is a numerical value which represents the amount of damage an entity can take before dying. Players and enemy NPCs have current and maximum health values which can both be changed. In multiplayer mode, players have their own health values individually. A player can revive another player when their health drops to zero.
- **Priority Level:** High
- **Possible Team Members:** Alex, Matias
- **Inputs:** Player health, enemy NPC health, damage dealt, healing received
- **Outputs:** Changes to player health and enemy NPC health
- **Process:**
    - All entities with health shall have a maximum health and current health represented by whole numbers where current health is initially set to the maximum health.
    - Entities that take damage shall decrease their current health by the amount of damage dealt until current health is zero. Once the current health is zero, the entity shall die
    - Interacting with healing objects shall increase the player's health up to the player's maximum health. Once interacted, the healing object shall disappear.
- **Dependencies:** Player interaction, player evade, enemy attacks, gameplay interface, enemy NPC stats

Player Upgrades

- **Description:** Player upgrades shall be found in levels. Players can interact with upgrades. Upgrades grant permanent increases to player health, damage, stun, and status effects. Upgrades are randomized and include upgrades any character can obtain and upgrades only specific characters can obtain.

- **Priority Level:** High
- **Possible Team Members:** Alex, Matias, Wyatt, Lance
- **Inputs:** Player inputs (keyboard, mouse, gamepad), player interact
- **Outputs:** Changes to player health, attack, skill, and status effects
- **Process:**
  - Players can find and interact with player upgrades within levels. Once interacted, players are given a selection from one to three randomly selected upgrades. Players can only choose one. Upgrades randomly include permanent increases to player health, attack, skill, damage, stun, and status effects with predetermined value increases.
  - In the first type of upgrade, the randomly selected upgrades are generated regardless of the character selected. In multiplayer, players can pick the same or different upgrades based on their gameplay preferences.
  - In the second type of upgrade, the randomly selected upgrades are specific to the character selected. In multiplayer, two sets of upgrades are generated, one for each of the player's characters.
- **Dependencies:** Player interact, player health, player attack, player skill, multiplayer

Currency

- **Description:** Defeated enemy NPCs shall drop currency that Players can pick up. Players shall interact with shop objects at the cost of currency. In multiplayer mode, the currency is shared among players.
- **Priority Level:** Medium
- **Possible Team Members:** Alex
- **Inputs:** Player collision, enemy NPC health, player interaction
- **Outputs:** Changes to player currency
- **Process:**
  - Defeated enemy NPCs shall drop a set number of currency objects. Currency shall move directly towards players in a nearby radius. Player collision with currency shall increase player's currency.
  - Shops have interactable objects that cost the player's currency. If players have sufficient currency, players can interact with shop objects which decreases player's currency by a set amount.
- **Dependencies:** Player collision, player interact, enemy NPCs, gameplay interface

Status Effects

- **Description:** Certain player and enemy NPC attacks will have the ability to apply various status effects to players, enhancing the complexity and challenge of combat encounters. Status effects will be categorized as damage-over-time (DoT), crowd control (CC), debuffs. These effects add strategic depth to the combat system, forcing players to react and adapt to various threats and allow unique ways to defeat enemies.

- **Priority Level:** Low
- **Possible Team Members:** Alex, Lance
- **Inputs:** Player attacks, enemy Attacks
- **Outputs:** A flexible system that allows for the easy application of status effects to both enemies and players. Status effects displayed on the player's UI, with timers or icons showing active effects. Visual and audio cues indicating when a status effect has been applied to a player or enemy.
- **Process:** Reusable Status Effect Framework: Develop a system where each status effect is treated as a modular component. This will allow easy application to any enemy NPC attack or environmental hazard and enable future additions of new effects without significant code changes.
    1. **Damage-over-Time Effects:** Implement periodic damage that scales based on time, dealing small amounts of damage at fixed intervals. These effects will be stackable, meaning the more times an enemy applies the effect, the greater the damage over time.
    2. **Crowd Control Effects:** Create a framework for canceling animations, temporarily immobilizing, or reducing the movement/attack speed of players or enemies
    3. **Debuff Effects:** Apply status changes that temporarily weaken the player or enemy stats. These debuffs will last for a fixed duration.
    4. **Triggering Mechanisms:** Implement rules for how status effects are triggered. Certain enemies may have attacks with a guaranteed status effect, while others have a random chance of applying one.
    5. **Status Indicators:** Ensure that whenever a status effect is applied, the player is clearly informed through UI elements, sound, and visual effects. A status bar or icon will indicate how long the effect will last.
- **Dependencies:** Player movement, player attack, player skill, player stats, enemy movement, enemy attacks, enemy stats, visual effects, gameplay interface

Enemy

Enemy Idle Movement

- **Description:** Enemies will move freely around their area whilst not pursuing the player. Grounded or walking enemies will only move along the ground and will not get stuck. Flying enemies will freely move in any direction and do not collide with platforms.
- **Priority Level:** High
- **Possible Team Members:** Lance, Alex
- **Inputs:** Enemy Position, Wall/Ledge Detection, Enemy Type

- **Outputs:** Change location, direction, and animation of enemy on screen.
- **Process:**
    1. **Ledge Detection:** Check if the enemy is near a wall or ledge (flying enemies ignore ledges).
    2. **Continue Patrolling:** If not, apply velocity in the direction that the enemy is facing and apply walking animation.
    3. **Switch Directions:** If so, apply velocity in the opposite direction that the enemy is facing and flip the direction of the sprite and animation.
- **Dependencies:** Player (hitbox), levels (walls and floors), sprites (animation), visual effects

Enemy Tracking

- **Description:** Enemy NPCs will follow unique tracking behaviors based on their type, reacting dynamically to the player's position within the game world. Using line-of-sight (LOS) algorithms, enemies will determine if they can detect the player and adjust their movement accordingly. Some enemies may be more aggressive, immediately chasing the player, while others may be more cautious or have specific behaviors when near obstacles such as walls or ledges.
- **Priority Level:** High
- **Possible Team Members:** Lance, Alex
- **Inputs:** Player Position, Enemy Type, Enemy Position, Wall/Ledge Detection
- **Outputs:** Change location, direction, and animation of enemy on screen with regards to the player's location.
- **Process:**
    1. **Player Detection:** Continuously track the Player Position in relation to the Enemy Position. If the player is outside the enemy's detection range or blocked from view, the enemy continues idle walking or patrol behavior based on its Enemy Type. If the player enters the detection range, check for Line of Sight (LOS). Perform LOS checks using raycasting or similar algorithms to determine if obstacles block the enemy's view of the player.
    2. **Tracking and Movement:** Based on Enemy Type, apply velocity toward or away from the Player Position. Different enemies will have different behaviors. Adjust the direction of the enemy's sprite and animation to match the direction of movement. Check for obstacles such as walls or ledges using wall/ledge detection triggers. If an enemy approaches a wall, it may either stop, jump, or climb depending on its behavior. If near a ledge, the enemy might stop or jump down based on predefined behaviors.
    3. **Animation Updates:** While tracking the player, update the enemy's animation to match its current state (e.g., idle, walking, running, climbing). Each state should have corresponding animations that switch dynamically during tracking.

4. **Reevaluate Player Position:** Continuously monitor the Player Position. If the player moves out of range or LOS is broken, the enemy will revert to its idle or patrol state. If the player remains within range, the enemy continues tracking and following.

- **Dependencies:** Player (position and hitbox), levels (walls and floors), sprites (animation), visual effects

Enemy Attacks

- **Description:** Enemy NPCs will engage players by performing unique attacks based on the enemy's type and proximity to the player. Each enemy will have a set of distinct attack patterns such as melee, ranged, and special abilities that can be directed at the player. The attack system will incorporate player tracking, attack animations, and status effects. Different enemy types will possess varying levels of intelligence and aggression, allowing for a dynamic combat experience.
- **Priority Level:** High
- **Possible Team Members:** Alex, Lance
- **Inputs:** Player Position, Enemy Type, Enemy Position, Status Effects
- **Outputs:** The enemy NPC performs an attack directed at the player, with an appropriate animation or visual effect (e.g., swing animation for melee, projectile animation for ranged attacks). If the attack lands, the system applies the corresponding effect (e.g., damage to player health, status effect application like poison or stun). Visual/audio feedback for the attack (e.g., sound effects, particle effects).
- **Process:**
    1. **Player Detection:** Continuously track the Player Position relative to the Enemy Position. If the player is outside the enemy's attack range, the enemy will continue tracking the player. If the player is within range, check for line of sight (Enemy Type Dependent). If no obstacles block the enemy's view of the player, proceed to attack.
    2. **Attack Preparation:** Depending on the Enemy Type, select an appropriate attack (e.g., melee, ranged, special ability). The enemy's behavior (e.g., aggression level, attack cooldown) will dictate when and how frequently they attack. Trigger attack animation or sprite change for the enemy NPC. For ranged enemies, spawn a projectile object directed at the player. For melee enemies, trigger a hitbox activation in the enemy's direction.
    3. **Attack Execution: Melee:** The enemy approaches the player and swings an attack. If the player is within the hitbox, apply damage and any additional status effects tied to the attack (e.g., poison). **Ranged:** The enemy launches a projectile. If the projectile collides with the player, apply damage and status effects.

    Special: Some enemies may cast spells or perform unique attacks that affect larger areas. These attacks will follow specific rules (e.g., area-of-effect zones, delayed explosions).

    4. **Attack Follow-Up:** After the attack, the enemy will either continue tracking the player or prepare for another attack based on its AI behavior (e.g., cooldown period, retreating, or moving to a new position).
    5. **Combat AI Adjustment:** The AI can adjust its tactics based on player movement or other game events (e.g., backing off after missing an attack, using cover in ranged combat, or entering an enraged state when low on health).
- **Dependencies:** Player (position and hitbox), levels (walls and floors), sprites (animation), visual effects

Boss Enemies

- **Description:** Players will face bosses at boss levels throughout their run. These bosses are designed with unique mechanics and behaviors that challenge the player's skill, making them difficult yet rewarding to defeat. The boss encounters will feature distinct attack patterns and phases that increase in intensity.
- **Priority Level:** Medium
- **Possible Team Members:** Wyatt, Nathaniel, Lance
- **Inputs:** Player Position, Player Health, Boss Health and State.
- **Outputs:** Three fully unique bosses, each with distinct mechanics, attack patterns, and phases.
- **Process:**
    1. **Boss Introduction:** When the player enters the boss arena (based on Player Position), a short cinematic or introductory animation will trigger, revealing the boss and starting the encounter.
    2. **Health Thresholds:** As the boss takes damage, specific health thresholds will trigger a change in behavior including new attacks and combos. Bosses may have no, one, or multiple health thresholds
    3. **Final Phase and Defeat:** Upon reducing the boss's health to zero, the boss will have a distinct defeat animation, and the player will be rewarded with player upgrades.
- **Dependencies:** Boss level, player (position and hitbox), sprites (animation), visual effects

<u>Levels</u>

Level Progression

- **Description:** Each level shall have an entrance and at least one or more exits which will lead to the next exploration or boss level. Players cannot trace back to previous levels.

Players in a run will experience multiple diverging and converging paths through exploration or boss levels. All paths end in the final boss level which is the final level concluding a run.

- **Priority Level:** High
- **Possible Team Members:** Nathaniel, Wyatt, Chester, Lance
- **Inputs:** Interaction with the exit of the level.
- **Outputs:** Change scene, load exploration or boss level
- **Process:**
    - Players will start a predetermined location at the start of every level.
    - At the end of the level, one or more exits will lead to the entrance of another level changing the scene. An exception is made for the final boss level where no exit is present and the run resets at the defeat of the final boss.
    - Generally, one or more exploration levels will be available to a player in between boss levels.
- **Dependencies:** Player interaction, exploration levels, boss levels

Exploration Levels

- **Description:** Exploration levels are large, randomly generated platforming stages consisting of rooms. Each unique level will follow a different set of rules for its level generation.
- **Priority Level:** Medium
- **Possible Team Members:** Nathaniel, Wyatt, Chester, Lance
- **Inputs:** Random Seed, Level generation rules
- **Outputs:** A randomly generated exploration level unique to each run and a corresponding map.
- **Process:**
    1. **Random Seed:** A seed will be randomly generated at the start of the run and stay the same until the run is reset
    2. **Level Generation:** After starting a new run or after leaving a previous level, the level will be randomly generated using the specific ruleset for the level and the seed. Level generation rules will define the layout of rooms, the types of rooms available to choose from, and the quantity of specific rooms
    3. **Rooms:** Rooms are hand-crafted subsections of exploration levels which offer a variety of experiences from platforming stages, enemy encounters, shops, upgrades, and story interactions.
- **Dependencies:** Map menu, enemy (spawn location), player upgrades (location placement)

Boss Levels

- **Description:** Boss levels are small static stages containing a boss enemy. They are handcrafted and consistent without any random generation. Special environmental objects interactable from the player or boss may be included in levels.
- **Priority Level:** Medium
- **Possible Team Members:** Nathaniel, Wyatt, Chester, Lance
- **Inputs:** Player or enemy interaction with environmental object
- **Outputs:** A boss level with a boss enemy and a corresponding map, environmental object effect
- **Process:**
    1. Boss stages have entrances and exits. When a player enters a boss stage, they will not be able to leave and take the exit until the boss is defeated.
    2. When a player or enemy interacts with an environmental object via input interact, movement, or attacks an environmental object effect is triggered
- **Dependencies:** Map menu, Boss Enemies, player (environmental object interaction), enemy (environmental object interaction)

Multiplayer

Local

- **Description:** Two players can play together on the same device using two inputs (keyboard/controller/combination of both).
- **Priority Level:** Medium
- **Possible Team Members:** Matias, Lance
- **Inputs:** Controller/Keyboard Inputs
- **Outputs:** Two player-controlled characters on the same screen with independent movement & actions. Co-op mechanics such as shared progression, reviving fallen teammates or sharing resources.
- **Process:**
    1. **Input Detection:** Detect two separate input sources (keyboard/controller/combination) when starting the game or entering a co-op mode.
    2. **Character Control:** Assign each input source to a different player character and map their respective controls (jump, attack, move, etc.).
    3. **Co-op Mechanics:** Implement shared mechanics such as assisting in combat, reviving, or getting upgrades together.
    4. **UI Adjustments:** Adjust the UI to support two players, ensuring both have clear health, skill, and inventory displays.
- **Dependencies:** Input System, UI System, Player Movement.

Online

- **Description:** Leveraging Unity's Gaming Services for multiplayer networking, players will connect via Unity's Relay and play together online in real-time, enabling cooperative gameplay. Players will be able to invite a friend for cooperative gameplay. Online play includes real-time synchronization of player actions, enemy movements, and shared progression.
- **Priority Level:** Low
- **Possible Team Members:** Matias, Lance
- **Inputs:** Player Input, Network Data
- **Outputs:** Real-time synchronized online gameplay between multiple players. Ability to invite a friend to play. Smooth and seamless cooperative experience using Unity's Networking and Relay services.
- **Process:**
    1. **Lobby Creation and Matchmaking:** Use Unity Gaming Services' Lobby API to allow players to create or join game sessions and invite a friend.
    2. **Network Synchronization:** Implement Unity's Netcode for GameObjects to synchronize player actions, combat, and movement in real-time across all connected clients.
    3. **Game State Synchronization:** Ensure game events like combat, item pickups, and level transitions are synchronized for all players.
    4. **Session Management:** Use Relay and Multiplay to manage session persistence, player disconnections, and rejoining scenarios.
    5. **Latency Optimization:** Use Unity's built-in Matchmaker and QoS (Quality of Service) features to optimize latency and maintain smooth gameplay across different regions.
- **Dependencies:** Unity Netcode for GameObjects, Unity Gaming Services(Lobby, Relay), Matchmaker and QoS.

Friend System

- **Description:** Players can manage a friends list within the game, allowing them to invite friends to join their online sessions easily. Using Unity's Gaming Services for social features, players can send and accept friend requests, see their friends' online status, and join or invite them into multiplayer games.
- **Priority Level:** Low
- **Possible Team Members:** Matias, Lance
- **Inputs:** Friend Requests, Friend List Data
- **Outputs:** A user interface displaying a friend list, online statuses, and game invites. Notifications for friend requests and multiplayer invitations.
- **Process:**

    1. **Friend List Management:** Use Unity's Friends and Social API to allow players to send friend requests and manage their friends list.
    2. **Online Status Notifications:** Show notifications when friends come online, start a session, or send an invite.
    3. **Game Invites and Joining:** Allow players to invite friends into a multiplayer session or join a session via invites through the Unity Gaming Services Lobby API.
    4. **UI Development:** Design a simple, intuitive UI to view friend lists, send invites, and manage requests within the game.
    5. **Backend Integration:** Store friend data and session history using Unity's backend services, ensuring persistence across player sessions.
- **Dependencies:** Unity Gaming Services (Friends, Social API), Lobby API

Database

Save Files

- **Description:** The Save Files Management system will handle player save files locally and synchronize them with Unity's Cloud Save services. Each save file will contain player-specific information such as unlocked characters, game progress, achievements, settings, and gameplay statistics. The feature will use local JSON files to save data on the player's device, ensuring smooth operation during offline play. Save files will be stored and managed both locally and in the cloud, allowing for easy restoration and synchronization when the player logs in on a different device or reconnects to the internet.
- **Priority Level:** High
- **Possible Team Members:** Chester, Lance, Nathaniel
- **Inputs:** Player Account Id, Game Progress, Settings & Preferences, Character Data, Achievements and Metadata.
- **Outputs:** A JSON file stored locally on the player's device. Updated cloud save data associated with the player's account in Unity Cloud Save. In-game display of available save files and respective metadata (e.g., playtime, completion status). Prompt messages for the player when a conflict arises between cloud and local save data.
- **Process:**
    1. **Create Save Files:** When the player starts a new game or creates a new save, generate a new JSON file in the local directory. Save all relevant data in this file and tag it with a unique identifier.
    2. **Load Save Files:** On game startup, check for available save files locally and in the cloud. If local files are found, populate the game's save file menu with metadata from these files. If cloud save files exist, attempt to synchronize and

   update the local files to reflect the most recent changes. Display a conflict resolution prompt if discrepancies are found between cloud and local save files.

3. **Save Game Progress:** Whenever the player makes significant progress (e.g., completes a level, unlocks a character), update the local JSON save file. Each time the save file is updated, mark it with a new timestamp for synchronization purposes. If the player is online, push the updated save data to the cloud.
4. **Sync with Cloud Save:** Use Unity's Cloud Save services to upload the save file data when connected to the internet. When the player logs into a new device, download the cloud save data and overwrite or merge with local save files as necessary. Keep track of the timestamps to ensure that the most recent data is always used.
5. **Conflict Resolution:** If a conflict is detected (e.g., the local file timestamp is newer than the cloud version), prompt the user to:
   Use Local Data: Overwrite the cloud data with local save data.
   Use Cloud Data: Overwrite the local data with the cloud version.
   Merge Data: Merge the most recent elements from both files.
6. **Delete Save Files:** Provide a way for players to delete save files locally and in the cloud. Ensure that once a file is deleted, it is removed from both local storage and cloud storage to prevent data mismatches.
7. **Backup and Restore:** Implement a local backup system where a copy of the save file is kept each time the game is saved. If a local file is corrupted or lost, the game can fall back on the most recent backup.

- **Dependencies:** Unity Cloud Save, UI System (for displaying save files and other prompts), JSON Parser, Backup system, game progress tracking system

Account

- **Description:** The account management system will handle both cloud and local storage to maintain user data. Accounts will be stored primarily in Unity's cloud database to keep track of global player progress and ensure data is synchronized across different devices. Locally, the account will also save essential information like user ID, username, game progress, unlocked characters, settings, and last login using a JSON file. This ensures that the game remains playable even when offline, and the data can be synced back to the cloud when the player reconnects. If discrepancies are found between cloud and local data, a conflict resolution mechanism will prompt the player to choose which data to keep.
- **Priority Level:** Medium
- **Possible Team Members:** Chester, Lance, Nathaniel
- **Inputs:** User login information (Username, Password) Local player data (JSON) Cloud player data (Unity Cloud Save) Device information (to handle local storage paths).

- **Outputs:** A unique user account is created, or existing account data is loaded. Local JSON data file (playerdata.json) is created or updated. Cloud account data is saved or synchronized with local data. Players can view or manage settings and account progress.
- **Process:**
    1. **Local Storage Implementation:** Define a PlayerData class to represent account data. Create a local JSON file (playerdata.json) in Unity's Application.persistentDataPath to store essential account data. Read from and write to this JSON file for quick access and offline functionality.
    2. **Cloud Storage Implementation:** Use Unity's CloudSaveService to manage online account data. Sync local data to the cloud after login or when the user finishes a game session. Handle cloud data using a similar structure, serializing and deserializing data in JSON format.
    3. **Login Process:** Prompt user for login credentials (username and password). Check if local data is present: If local data is present and no internet connection, load data from the local JSON file. If both local and cloud data are available, sync the data using the timestamp or ask the user to choose. If only cloud data is present, download the data and create a local backup. Display the main menu with the player's data.
    4. **Account Data Synchronization:** On game start, attempt to sync the local data with the cloud. If a player makes changes (e.g., unlocks a character), update both local and cloud data. When the player logs out or exits the game, push all changes to the cloud to ensure no data loss.
    5. **Conflict Resolution:** If there is a mismatch between the cloud and local data, prompt the player with: "Use Local Data" (keep current device's data) "Use Cloud Data" (sync with cloud and overwrite local) "Merge Data" (combine aspects of both).
- **Dependencies:** Unity Cloud Save for database management. Local JSON storage for offline functionality. User authentication through Unity Authentication services (for managing player sessions).

Run Information

- **Description:** The Run Information feature tracks and stores detailed data for each individual run (game session) that a player completes, such as the total duration, enemies defeated, damage taken, items collected, rooms explored, bosses defeated, and more. Each run's data will be stored locally in a JSON format and synchronized with Unity's Cloud Save service for leaderboard integration and analytical tracking. This system allows players to review their performance and compare runs, while also enabling in-game achievements and progression tracking based on performance metrics.
- **Priority Level:** Medium
- **Possible Team Members:** Chester, Lance, Nathaniel

- **Inputs:** Player Actions, Player Position, Time Data, In-Game Events, Player Stats
- **Outputs:** A detailed JSON file that records every aspect of the run. Aggregated performance data, such as total playtime, number of deaths, highest damage dealt, or items found. A visual summary of the run shown to the player upon death or completion, including stats like "Fastest Run" or "Most Enemies Defeated." Data synchronized with the cloud for leaderboard submission and achievements.
- **Process:**
    1. **Track Run Events:** During gameplay, continuously track all major events such as entering a new room, defeating enemies, using items, and taking damage. Store key data points in a local run log.
    2. **Update Run Data:** As players progress, the system updates real-time information, adding to the cumulative stats for the current run. Track timestamps for key events (e.g., entering/exiting rooms, starting/ending boss fights) to generate detailed insights.
    3. **End Run and Save Data:** When the player dies or completes the run, finalize the RunData object by setting the end time and final outcomes. Convert the RunData object into a JSON file and save it locally. Immediately sync the data with Unity's Cloud Save services if the player is online.
    4. **Sync with Cloud Services:** Upload the run data to Unity's Cloud Save to update the player's profile and leaderboard stats. Data can be aggregated to show a player's highest scores, longest survival times, and other metrics in leaderboards or player profiles.
    5. **Display Run Summary:** Upon completion of a run, display a detailed summary of the run to the player. Include information such as total enemies defeated, run time, items found, bosses defeated, and a comparison with previous runs. Highlight notable achievements like "Fastest Run" or "Most Enemies Defeated" to reward player performance.
    6. **Leaderboard Integration:** Use Unity Gaming Services' leaderboard integration to submit run data for ranking players based on their performance. Compare the player's run data to the top players or friends to encourage competition.
    7. **Data Retention and Clean-Up:** Implement a local retention policy to only keep a limited number of the most recent runs locally to avoid bloating the device storage. Sync all data to the cloud before deletion, ensuring a complete backup is maintained.
- **Dependencies:** Local Save System (JSON) Unity Cloud Save, Unity Leaderboards, User Account System, UI System.

Visual

Camera

- **Description:** In this 2D JRPG side-scroller, the camera should follow the player smoothly, usually centered on the character with some slight lag for a more dynamic feel. It should limit vertical movement to keep the focus on the horizontal plane, and include boundaries to prevent showing off-screen areas.
- **Priority Level:** High
- **Possible Team Members:** Wyatt, Nathaniel
- **Inputs:** The scene or setting of the player, the player(s),
- **Outputs:** Visual ease of player(s) location, enemies location, item location, and location of all other interactables such as doors and any NPCs (Non-Playable Characters) as players traverse the map.
- **Process:**
    - Following movement and location of the player(s)
    - Making sure the player(s) are in frame of gameplay
    - Smooth flow through environment as players and enemies pass through
    - Smooth flow horizontally through level environment and obstacles
    - Smooth flow vertically as players descend and ascend throughout the levels environment
- **Dependencies:** Player Positioning, Level Design, Camera Boundaries, Zoom and Scaling, Camera Shake, Parallax Scrolling, Object Tracking, Input Responsiveness, Events and Triggers, Lighting, VFX, Frame Rate, UI Integration

Sprites

- **Description:** Sprites in this 2D JRPG should be vibrant and expressive, often featuring pixel art or hand-drawn styles that reflect the game's aesthetic. Character sprites typically include multiple frames for smooth animations, such as walking, attacking, and idle poses. They should convey emotions through facial expressions and poses. Environmental sprites, like trees and buildings, should match the art style and create a cohesive world for the player.
- **Priority Level:** High
- **Possible Team Members:** Wyatt, Nathaniel
- **Inputs:** movement, actions, abilities, camera and menu navigation, button feedback, interactive actions with objects and other sprites or NPCs (Non-Playable Characters)
- **Outputs:** A functional character with smooth like animation that feels immersive to the player and their experience when playing a 2D JRPG. Enemies, players, and bosses are distinguishable.
- **Process:**
    - Unique artistic representation of the character, bosses, and enemies
    - Visual clarity between characters, enemies, and bosses
    - Animated movement frames

- **Dependencies:** Animation State, Collision Detection, Sprite Sheet, Layering and Depth, Transformations (sprite facing left or right), Physics Integration, Input and Controls, Hitboxes and Hurtboxes, Visual Effects, UI Interaction, Sound Effects, Design Consistency, Environment Interactions, Performance Optimization

VFX

- **Description:** Visual effects will consist of all action animations, such as attacks and movement, and level of lighting on stages, as well as for the player and enemies. VFX will include particle effects (like fire, smoke, and explosions), animated textures, screen effects (like blurs or glows), environmental effects (like rain or snow), and UI effects (like button animations). These elements work together to enhance the visual experience and immerse the users in the 2D world.
- **Priority Level:** Low
- **Possible Team Members:** Wyatt, Nathaniel
- **Inputs:** Particle effects, textures, lighting, environment, UI effects
- **Outputs:** immersive unique visuals from abilities, characters, UI, and the environment.
- **Process:**
  - Trigger VFX for player attacks and movement
  - Trigger VFX during enemy actions, such as special attacks or movement, and synchronize with their animations
  - generate weather effects (rain, snow) based on the level's environment.
  - Activate particle effects (such as explosions, fire, smoke) when specific actions occur, such as during attacks, item use, or environmental interactions.
- **Dependencies:** Sprite Integration, Particle Systems, Timing and Animation, Layering and Depth, Lighting and Shadows, Color Palettes, Performance Optimization, Sound Integration, Camera Interaction, Trigger Systems, Environment Effects, UI Feedback, Customization (skills and item VFX adapting to change)

Audio

Music

- **Description:** In this 2D JRPG side-scroller, music should enhance the action and pacing, often using a mix of upbeat tunes for exploration and more intense themes for combat. Each level may feature unique tracks that reflect the environment, with layered melodies to create depth. Transitioning music smoothly as players move between areas keeps the experience immersive.
- **Priority Level:** Low
- **Possible Team Members:** Wyatt, Matias
- **Inputs:** background, battle, event triggered, cutscene, credits, title screen, audio files

- **Outputs:** background music for levels and combat sequences, smooth transition between exploration, combat, and boss battle music, event based music changes
- **Process:**
    - Upon entering an area, the system selects the appropriate background track based on the level's theme.
    - Smooth transitions are implemented when moving between different gameplay states (exploration -> combat -> boss).
- **Dependencies:** Layering, Looping Mechanism, Environmental Context, Character Theme, Sound Design Integration, Adaptive Systems (low health or combat status), Mood and Tone, Event Triggers, Cutscene Integration, Accessibility Considerations, Feedback Mechanism (health warnings or special events), Localization

Sound FX

- **Description:** Sound effects play a vital role, adding impact to actions and interactions. Sound effects will include character actions, environmental sounds and ambiance, UI sounds, combat effects, and narrative cues, such as dialogue and story.
- **Priority Level:** Low
- **Possible Team Members:** Wyatt, Matias
- **Inputs:** audio files, combat actions, interactable items, environment
- **Outputs:** Sounds will immerse the player, enhancing the gaming experience with unique effects for players, enemies, objects, and environment.
- **Process:**
    - Environmental sounds are triggered based on specific events, such as opening doors, activating traps.
    - When the player performs an action (attack, jump, etc..), corresponding sound effects are played based on the type of action.
- **Dependencies:** Emotional Cues, Pacing and Timing, Accessibility Features, Localization, Performance Optimization, Differing Soundscapes, Layering, Trigger Systems, UI Sounds, Environment Ambiance, Enemy Sounds, Item Collection, Combat Feedback, Environment Interactions, Character Actions

Non-Functional Requirements:

<u>User Interface & Visual</u>

Game Screen

- **Response Time:**
    - The total screen refresh rate (frame rate) should not drop below 30 frames per second for all screens displayed to players

    - All players, enemies, entities, and ui elements shown on screen shall be updated in engine at least 30 times per second

<u>Player</u>

Player Controls

- **Usability:**
    - Players should be able to use either mouse and keyboard or controller to control characters.
    - Players should be able to switch between types of player controls
    - Local multiplayer players should be able to change and update the controller they are using to control their respective characters
- **Response Time:**
    - The total response time from player controller, to the server, and to the screen for all players should not exceed 300 ms

<u>Gameplay</u>

Crash and Deadlock

- **Reliability:**
    - The player shall not crash more than once per run
    - The player shall not encounter a deadlock where players are unable to progress the game
        - Progress is the ability for players to exit levels, enter next levels, defeat boss enemies, and end the run

<u>Levels</u>

Level Transitions

- **Response Time:**
    - The total loading time between levels should not exceed 30 seconds to procedurally generate and load all players, enemies, and elements into scene
- **Reliability:**
    - Player shall be able to exit every level from the beginning of the game until the end without deadlock

<u>Multiplayer</u>

Online Multiplayer

- **Reliability:**
    - Should one player crash or disconnect, the other player shall not disconnect and be able to continue the game in single player

- **Response Time:**
    - The total loading time from accept invite to joining a player's lobby should be no longer than 30 seconds

<u>Data</u>

Account

- **Security:**
    - Accounts shall be authenticated with password protection
    - Password information shall be inaccessible to users on the local computer or through the server

Hardware Limitations

- **Constraints:**
    - OS: Windows 10
    - Processor: Intel Core i5
    - Memory: 8 gb Ram
    - Graphics: GeForce GTX 1650
    - Storage: 10 gb
        - Shall not exceed limit on local computer
- **Efficiency**
    - Memory usage shall not exceed 4 gb Ram

Save File & Run Information

- **Security:**
    - Save files and run information should not be manipulable by players on local computer
- **Reliability:**
    - Save files and run information should be updated at least once per level for all players